15-213: Final Exam Review

Minji, Harsha, and Ishita
Threads and Synchronization

INTERVIEWER: EXPLAIN WHAT A DEADLOCK IS AND THE JOB IS YOURS

15-213 STUDENT: HIRE ME FIRST AND I WILL EXPLAIN TO YOU WHAT A DEADLOCK IS.
Threads and Synchronization

Problem Statement:

- 15-213 TAs now want to begin a new procedure for daily office hours:

- When a student enters the room, he/she will see a table inside the room with several pens and several notebooks on it (if the pens and notebooks haven't been claimed by other students). TAs are inside the room waiting to help them.
Threads and Synchronization

Procedure for students:

■ Wait until you get called, claim a pen and a notebook.
■ Use the pen to write your questions on the notebook.
■ After you are done with question writing, release the pen.
■ Hold the notebook and wait until a TA is free to help you.
■ After you are done with talking to a TA about the question, go back to claim a pen and write the solutions on the notebook.
■ Release the pen and the notebook and go home.
Typical Code to launch threads and wait for them to finish:

```c
sem_t pen;
sem_t notebook;
sem_t ta;

int main(int argc, char** argv)
{
    int P = 5, N = 5, T = 3, S = 50;
    sem_init(&pen, 0, P);
    sem_init(&notebook, 0, N);
    sem_init(&ta, 0, T);

    int* students = malloc(S * sizeof(int));
    launch_threads();
    reap_threads();
}

void launch_threads()
{
    for(int i = 0; i < S; i++)
    {
        students[i] =
        pthread_create(students+i,
                        NULL, student_fn, NULL);
    }
}

void reap_threads()
{
    for(int i = 0; i < S; i++)
    {
        pthread_join(students[i]);
    }
}
```
void* student_fn(void* varp)
{
    P(&pen);
    P(&notebook);
    sleep(genrand() % 60); // Time for you to write questions
    V(&pen);
    P(&ta);
    sleep(genrand() % 900); // Time for you to ask TA
    V(&ta);
    P(&pen);
    sleep(genrand() % 60); // Time for you to write solutions
    V(&notebook);
    V(&pen);
    return 0;
}

However, the code sometimes gets stuck by a deadlock. Which of the following is correct?

- A: The deadlock is caused by the pen and the notebook.
- B: The deadlock is caused by the notebook and the TA.
- C: The deadlock is caused by the TA and the pen.
Threads and Synchronization

A student writes the following program to simulate the given procedures

```c
void* student_fn(void* varp) {
    P(&pen);
    P(&notebook);
    sleep(genrand() % 60); // Time for you to write questions

    V(&pen);
    P(&ta);
    sleep(genrand() % 900); // Time for you to ask TA

    V(&ta);
    P(&pen);
    sleep(genrand() % 60); // Time for you to write solutions

    V(&notebook);
    V(&pen);

    return 0;
}
```

Let us consider a simple example:

- Pens: 5
- Notebooks: 5
- TAs:
- Students: 50
void* student_fn(void* varp) {
    P(&pen);
    P(&notebook);
    sleep(genrand() % 60); // Time for you to write questions
    V(&pen);
    P(&ta);
    sleep(genrand() % 900); // Time for you to ask TA
    V(&ta);
    P(&pen);
    sleep(genrand() % 60); // Time for you to write solutions
    V(&notebook);
    V(&pen);
    return 0;
}

Let's analyse the semaphores

Case C: TAs and Pens

- There is no overlap
- There can't be a case where a student who is talking to a TA is waiting on a pen.
- There can't be a case where a student who is holding a pen is waiting on a TA
void* student_fn(void* varp)
{
    P(&pen);
    P(&notebook);
    sleep(genrand() % 60); // Time for you to write questions
    V(&pen);
    P(&ta);
    sleep(genrand() % 900); // Time for you to ask TA
    V(&ta);
    P(&pen);
    sleep(genrand() % 60); // Time for you to write solutions
    V(&notebook);
    V(&pen);
    return 0;
}

Let’s analyse the semaphores

Case B: TAs and Notebooks

- There is an overlap
- A student who is holding a notebook may be waiting on a TA
- But….A student who is talking to a TA will not be waiting on a notebook
void* student_fn(void* varp)
{
    P(&pen);
P(&notebook);
sleep(genrand() % 60); // Time for you to write questions
    V(&pen);
P(&ta);
sleep(genrand() % 900); // Time for you to ask TA
    V(&ta);
P(&pen);
sleep(genrand() % 60); // Time for you to write solutions
    V(&notebook);
    V(&pen);
return 0;
}

Case A: Pens and Notebooks

- There is an overlap
- A student who is holding a notebook may be waiting on a pen
- A student who is holding a pen may be waiting on a notebook

Let's analyse the semaphores

W: Get a pen
S: Give up pen
W: Get a TA
S: Give up TA
W: Get a pen
S: Give up pen
S: Signal
W: Wait
W: Get a Notebook
S: Give up Notebook
Threads and Synchronization

Let’s analyse the semaphores

```c
void* student_fn(void* varp)
{
    P(&pen);
    P(&notebook);

    sleep(genrand() % 60); // Time for you to write questions
    V(&pen);
    P(&ta);
    sleep(genrand() % 900); // Time for you to ask TA
    V(&ta);
    P(&pen);
    sleep(genrand() % 60); // Time for you to write solutions
    V(&notebook);
    V(&pen);

    return 0;
}
```

Case A: Pens and Notebooks
Consider this likely scenario:

- 5 Students (group A) enter the room
- They grab all 5 pens and all 5 notebooks on the table.
- Every student behind them is waiting on a pen and a notebook. (Group B)
- All 5 students in Group A give up their pens, but not their notebooks.
- 5 students in Group B immediately grab the 5 pens.
- Group B are now waiting on notebooks.
- After talking to the TAs, all students in Group A are now waiting on pens, which students in Group B have.
- Group B is waiting on the notebooks which students in Group A have.

DEADLOCK!!
void* student_fn(void* varp) {
    P(&pen);
    P(&notebook);

    sleep(genrand() % 60); // Time for you to write questions

    V(&pen);
    P(&ta);

    sleep(genrand() % 900); // Time for you to ask TA

    V(&ta);
    P(&pen);

    sleep(genrand() % 60); // Time for you to write solutions

    V(&notebook);
    V(&pen);

    return 0;
}

However, the code sometimes gets stuck by a deadlock. Which of the following is correct?

A: The deadlock is caused by the pen and the notebook.

B: The deadlock is caused by the notebook and the ta.

C: The deadlock is caused by the ta and the pen.

ANSWER: A
Threads and Synchronization

Can we fix this without changing the Office Hour procedure?
Threads and Synchronization

Can we fix this without changing the Office Hour procedure?

YES!
Let’s analyse the semaphores

```c
void* student_fn(void* varp)
{
    P(&notebook);
    P(&pen);

    sleep(genrand() % 60); // Time for you to write questions

    V(&pen);
    P(&ta);

    sleep(genrand() % 900); // Time for you to ask TA

    V(&ta);
    P(&pen);

    sleep(genrand() % 60); // Time for you to write solutions

    V(&pen);
    V(&notebook);

    return 0;
}
```

Let’s just re-order the locks and analyse it again.

- A student who is holding a notebook may wait on a pen or a TA

However...

- A student who is holding a pen will already have a notebook (and never wait on it).
- A student who is talking to a TA will already have a notebook (and never wait on it).
Signals

who would win?

several hundred lines of tshlab code

one asynchronous boi

```c
void exec_cmdline(char *cmdline, char **argv, sigset_t *set, int bg, int fd_in, int fd_out)
{
    pid_t pid = 0;
    if ((pid = Fork()) == 0) {
        // Child process; restore mask and execute job.
        Sigprocmask(SIG_SETMASK, set, NULL);
    }
}
```
Examples in these slides are important “gotchas”
- NOT an all inclusive list
- In particular, these do NOT go over signals sent between multiple processes

Many more examples in:
- Recitation slides
- Lecture slides
- Past exams

Reminder: solve the questions on your own when studying before looking at the answer
Does the following code ever terminate?

```c
void handler(int sig) {
    while(1);
}

int main() {
    Signal(SIGUSR1, handler);
    Kill(0, SIGUSR1);
    return 0;
}
```
Signals

- Does the following code ever terminate?

```c
void handler(int sig) {
    while(1);     // stuck here forever!
}

int main() {
    Signal(SIGUSR1, handler);
    // Spec says signal **sent to self** must be **handled**
    // **before** kill() **returns**
    Kill(0, SIGUSR1);
    return 0;
}
```
Signals

What about now?

```c
void handler(int sig) {
    Kill(0, SIGUSR1);
}

int main() {
    Signal(SIGUSR1, handler);
    Kill(0, SIGUSR1);
    return 0;
}
```
Signals

What about now?

```c
void handler(int sig) {
    Kill(0, SIGUSR1);
}
```

```c
int main() {
    Signal(SIGUSR1, handler);
    Kill(0, SIGUSR1);
    return 0;
}
```

Does not terminate:

```c
main: Kill(0, SIGUSR1);
<SIGUSR1 handler invoked, SIGUSR1 blocked in handler>
handler: Kill(0, SIGUSR1);
<SIGUSR1 pending>
<handler returns>
<SIGUSR1 unblocked, SIGUSR1 handler invoked, SIGUSR1 blocked in handler>
handler: Kill(0, SIGUSR1);
...repeat...
```
Processes

REAP

ALL THE ZOMBIE CHILDREN!
Processes

Draw a Process Graph!!!

(it does not have to be like mine)
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
            count += 2;
    }
    printf("%d", count);
}
Processes

```c
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
            count += 2;
    }
    printf("%d", count);
}
```

How many processes?
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;  
        else
            count += 2;
    }
    printf("%d", count);
}
Processes

```c
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
            count += 2;
    }
    printf("%d", count);
}
```

What does the process diagram look like?
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
            count += 2;
    }
    printf("%d", count);
}
```c
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
            count += 2;
    }
    printf("%d", count);
}
```

What does count look like?

Parent: pid1 != 0 and pid2 != 0
Child1: pid1 == 0 and pid2 != 0
Child2: pid1 != 0 and pid2 == 0
Grandchild: pid1 == 0 and pid2 == 0
Processes

```c
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
            count += 2;
    }
    printf("%d", count);
}
```

What does count look like?

- **Parent**: `pid1 != 0 and pid2 != 0`
  - `count = 3`
- **Child1**: `pid1 == 0 and pid2 != 0`
  - `count = 2`
- **Child2**: `pid1 != 0 and pid2 == 0`
  - `count = 0`
- **Grandchild**: `pid1 == 0 and pid2 == 0`
  - `count = 2`
Given the process diagram, what are the different permutations that can be printed out?

```c
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
            count += 2;
    }
    printf("%d", count);
}
```
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;  
        else
            count += 2;
    }
    printf("%d", count);
}

Given the process diagram, what are the different permutations that can be printed out?

Math! $\frac{4!}{2} = 12$ different possible outcomes
Processes

Remember:

● Processes can occur in any order
● Watch out for a wait or a waitpid!
  ○ What if I included a wait(NULL) before I printed out count?
● Good luck!
File IO

.DAT FILE
int main() {
    int fd1, fd2, fd3;
    char c;
    pid_t pid;
    fd1 = open("foo.txt", O_RDONLY);
    fd2 = open("foo.txt", O_RDONLY);
    fd3 = open("foo.txt", O_RDONLY);
    read(fd1, &c, sizeof(c));  // c = ?
    read(fd2, &c, sizeof(c));  // c = ?
    dup2(fd2, fd3);
    read(fd3, &c, sizeof(c));  // c = ?
    read(fd2, &c, sizeof(c));  // c = ?

Main ideas:
■ How does read offset?
■ How does dup2 work?
  ■ What is the order of arguments?
  ■ Does fd3 share offset with fd2?
Carnegie Mellon

foo.txt: abcdefgh...xyz
int main() {
    int fd1, fd2, fd3;
    char c;
    pid_t pid;
    fd1 = open("foo.txt", O_RDONLY);
    fd2 = open("foo.txt", O_RDONLY);
    fd3 = open("foo.txt", O_RDONLY);
    read(fd1, &c, sizeof(c)); // c = a
    read(fd2, &c, sizeof(c)); // c = a
    dup2(fd2, fd3);
    read(fd3, &c, sizeof(c)); // c = b
    read(fd2, &c, sizeof(c)); // c = c

    ■ How does read offset?
        ■ Incremented by number of bytes read
    ■ How does dup2 work?
        ■ Any read/write from fd3 now happen from fd2
        ■ All file offsets are shared

File IO
read(fd1, &c, sizeof(c)); // a
read(fd2, &c, sizeof(c)); // a
dup2(fd2, fd3);
read(fd3, &c, sizeof(c)); // b
read(fd2, &c, sizeof(c)); // c

pid = fork();
if (pid==0) {
    read(fd1, &c, sizeof(c));
    printf("c = %c\n", c);
    dup2(fd1, fd2);
    read(fd3, &c, sizeof(c));
    printf("c = %c\n", c);
}
read(fd2, &c, sizeof(c));
printf("c = %c\n", c);
read(fd1, &c, sizeof(c));
printf("c = %c\n", c);

Main ideas:

- How are fd shared between processes?
- How does dup2 work from parent to child?
- How are file offsets shared between processes?
read(fd1, &c, sizeof(c)); // a
read(fd2, &c, sizeof(c)); // a
dup2(fd2, fd3);
read(fd3, &c, sizeof(c)); // b
read(fd2, &c, sizeof(c)); // c

pid = fork();
if (pid==0) {
    read(fd1, &c, sizeof(c));
    printf("c = %c\n", c);
    dup2(fd1, fd2);
    read(fd3, &c, sizeof(c));
    printf("c = %c\n", c);
}
read(fd2, &c, sizeof(c));
printf("c = %c\n", c);
read(fd1, &c, sizeof(c));
printf("c = %c\n", c);
read(fd1, &c, sizeof(c)); // a
read(fd2, &c, sizeof(c)); // a
dup2(fd2, fd3);
read(fd3, &c, sizeof(c)); // b
read(fd2, &c, sizeof(c)); // c

pid = fork();
if (pid==0) {
    read(fd1, &c, sizeof(c));
    printf(“c = %c\n”, c);
    dup2(fd1, fd2);
    read(fd3, &c, sizeof(c));
    printf(“c = %c\n”, c);
}
read(fd2, &c, sizeof(c));
printf(“c = %c\n”, c);
read(fd1, &c, sizeof(c));
printf(“c = %c\n”, c);

Possible output 1:

  c = d // in parent
  c = b // in parent
  c = c // in child from fd1
  c = e // in child from fd3
  c = d // in child
  c = e // in child
File IO

```c
read(fd1, &c, sizeof(c)); // a
read(fd2, &c, sizeof(c)); // a
dup2(fd2, fd3);
read(fd3, &c, sizeof(c)); // b
read(fd2, &c, sizeof(c)); // c

pid = fork();
if (pid==0) {
    read(fd1, &c, sizeof(c));
    printf("c = %c\n", c);
    dup2(fd1, fd2);
    read(fd3, &c, sizeof(c));
    printf("c = %c\n", c);
}
read(fd2, &c, sizeof(c));
printf("c = %c\n", c);
read(fd1, &c, sizeof(c));
printf("c = %c\n", c);
```

Possible output 2:

- `c = b // in child`
- `c = d // in child`
- `c = c // in child`
- `c = d // in child`
- `c = e // in parent`
- `c = e // in parent`
pid = fork();
if (pid==0) {
    read(fd1, &c, sizeof(c));
    printf("c = %c\n", c);
    dup2(fd1, fd2);
    read(fd3, &c, sizeof(c));
    printf("c = %c\n", c);
}
if (pid!=0) waitpid(-1, NULL, 0);
read(fd2, &c, sizeof(c));
printf("c = %c\n", c);
read(fd1, &c, sizeof(c));
printf("c = %c\n", c);
return 0;

What are the possible outputs now?
```c
pid = fork();
if (pid==0) {
    read(fd1, &c, sizeof(c));
    printf("c = %c\n", c);
    dup2(fd1, fd2);
    read(fd3, &c, sizeof(c));
    printf("c = %c\n", c);
}
if (pid!=0) waitpid(-1, NULL, 0);
read(fd1, &c, sizeof(c));
printf("c = %c\n", c);
read(fd2, &c, sizeof(c));
printf("c = %c\n", c);
return 0;
```

Possible output:

```
c = b // in child
c = d // in child
c = c // in child
c = d // in child
c = e // in parent
c = e // in parent
```
read(fd1, &c, sizeof(c)); // a
read(fd2, &c, sizeof(c)); // a
dup2(fd2, fd3);
read(fd3, &c, sizeof(c)); // b
read(fd2, &c, sizeof(c)); // c

pid = fork();
if (pid==0) {
    read(fd1, &c, sizeof(c));
dup2(fd1, fd2);
    read(fd3, &c, sizeof(c));
} else { if (pid!=0) waitpid(-1, NULL, 0);
read(fd2, &c, sizeof(c));
read(fd1, &c, sizeof(c));

- Child creates a copy of the parent fd table
- dup2/open/close in parent only before fork() affect the child
- dup2/open/close in child do NOT affect the parent
- File descriptors across process share the same file offset.
Virtual Memory

When did you become an expert in virtual memory?

Last night
Virtual Memory

Virtual Address - 18 Bits

Physical Address - 12 Bits

Page Size - 512 Bytes

TLB is 8-way set associative

Cache is 2-way set associative

Final S-02 (#5)

Lecture 18: VM - Systems
Virtual Memory

Label the following:
(A) VPO: Virtual Page Offset
(B) VPN: Virtual Page Number
(C) TLBI: TLB Index
(D) TLBT: TLB Tag
Virtual Memory

Label the following:

(A) **VPO:** Virtual Page Offset - Location in the page
    Page Size = 512 Bytes = $2^9 \rightarrow$ Need 9 bits
Virtual Memory

Label the following:

(A) VPO: Virtual Page Offset
(B) VPN: Virtual Page Number - Everything Else
Virtual Memory

Label the following:

(A) **VPO**: Virtual Page Offset
(B) **VPN**: Virtual Page Number
(C) **TLBI**: TLB Index - Location in the TLB Cache

```
17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
B B B B B B B B B B A A A A A A A A A A
```
Virtual Memory

Label the following:

(A) **VPO:** Virtual Page Offset
(B) **VPN:** Virtual Page Number
(C) **TLBI:** TLB Index - Location in the TLB Cache

2 Indices → 1 Bit
Virtual Memory

Label the following:

(A) VPO: Virtual Page Offset
(B) VPN: Virtual Page Number
(C) TLBI: TLB Index
(D) TLBT: TLB Tag - Everything Else
Virtual Memory

Label the following:

(A) $PPO$: Physical Page Offset
(B) $PPN$: Physical Page Number
(C) $CO$: Cache Offset
(D) $CI$: Cache Index
(E) $CT$: Cache Tag
Virtual Memory

Label the following:

(A)  *PPO*: Physical Page Offset
Virtual Memory

Label the following:

**(A) PPO:** Physical Page Offset - Same as VPO
Virtual Memory

Label the following:

(A) \textit{PPO}: Physical Page Offset - Same as VPO
(B) \textit{PPN}: Physical Page Number - Everything Else
Virtual Memory

Label the following:

(A)  **PPO**: Physical Page Offset - Same as VPO

(B)  **PPN**: Physical Page Number - Everything Else

(C)  **CO**: Cache Offset - Offset in Block
Virtual Memory

Label the following:

(A) **PPO**: Physical Page Offset - Same as VPO
(B) **PPN**: Physical Page Number - Everything Else
(C) **CO**: Cache Offset - Offset in Block

4 Byte Blocks → 2 Bits
Virtual Memory

Label the following:

(A) **PPO**: Physical Page Offset - Same as VPO
(B) **PPN**: Physical Page Number - Everything Else
(C) **CO**: Cache Offset - Offset in Block
(D) **CI**: Cache Index

![Diagram of memory allocation]

- B B B A A A A A A A A A A
- CO
Virtual Memory

Label the following:

(A) **PPO**: Physical Page Offset - Same as VPO

(B) **PPN**: Physical Page Number - Everything Else

(C) **CO**: Cache Offset - Offset in Block

(D) **CI**: Cache Index

4 Indices $\rightarrow$ 2 Bits
Virtual Memory

Label the following:

(A) **PPO**: Physical Page Offset - Same as VPO
(B) **PPN**: Physical Page Number - Everything Else
(C) **CO**: Cache Offset - Offset in Block
(D) **CI**: Cache Index
(E) **CT**: Cache Tag - Everything Else
Virtual Memory

Now to the actual question!

Q) Translate the following address: 0x1A9F4
Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation

   - 1 = 0001
   - A = 1010
   - 9 = 1001
   - F = 1111
   - 4 = 0100

   ![Bit Representation Image]

   0 1 1 0 1 0 1 0 0 1 1 1 1 1 1 0 1 0 0 0
Virtual Memory

Now to the actual question!

Q) Translate the following address: 0x1A9F4

1. Write down bit representation
2. Extract Information:
   VPN: 0x??      TLBI: 0x??      TLBT: 0x??
   TLB Hit: Y/N?  Page Fault: Y/N?  PPN: 0x??
Virtual Memory

Now to the actual question!

Q) **Translate the following address**: 0x1A9F4

1. Write down bit representation
2. Extract Information:
   - VPN: 0xD4
   - TLBI: 0x??
   - TLBT: 0x??
   - TLB Hit: Y/N?
   - Page Fault: Y/N?
   - PPN: 0x??
Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information:
   - VPN: 0xD4
   - TLBI: 0x00
   - TLBT: 0x??
   - TLB Hit: Y/N?
   - Page Fault: Y/N?
   - PPN: 0x??
Virtual Memory

Now to the actual question!

Q) Translate the following address: 0x1A9F4

1. Write down bit representation
2. Extract Information:
   - VPN: 0xD4
   - TLBI: 0x00
   - TLBT: 0x6A
   - TLB Hit: Y/N?
   - Page Fault: Y/N?
   - PPN: 0x??

Binary representation: 011101010001000111111111010
Virtual Memory

Now to the actual question!

Q) Translate the following address: 0x1A9F4

1. Write down bit representation
2. Extract Information:
   - VPN: 0xD4
   - TLBI: 0x00
   - TLBT: 0x6A
   - TLB Hit: Y!
   - Page Fault: Y/N?
   - PPN: 0x??
Virtual Memory

Now to the actual question!

Q) **Translate the following address:** 0x1A9F4

1. Write down bit representation
2. Extract Information:
   - VPN: 0xD4
   - TLBI: 0x00
   - TLBT: 0x6A
   - TLB Hit: Y!
   - Page Fault: N!
   - PPN: 0x??
Virtual Memory

Now to the actual question!

Q) Translate the following address: 0x1A9F4

1. Write down bit representation
2. Extract Information:
   - VPN: 0xD4
   - TLBI: 0x00
   - TLBT: 0x6A
   - TLB Hit: Y!
   - Page Fault: N!
   - PPN: 0x3

```
0 1 1 1 0 1 0 1 0 0 0 1 1 1 1 1 1 1 0 1 0 0
```
Virtual Memory

Now to the actual question!

Q) Translate the following address: 0x1A9F4

1. Write down bit representation
2. Extract Information
3. Put it all together: PPN: 0x3, PPO = 0x??
Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information
3. Put it all together: PPN: 0x3, PPO = VPO = 0x1F4
Virtual Memory

Q) What is the value of the address?

CO: 0x??  CI: 0x??  CT: 0x??  Cache Hit: Y/N?  Value: 0x??
Virtual Memory

Q) **What is the value of the address?**
1. Extract more information

CO: 0x00   CI: 0x??   CT: 0x??   Cache Hit: Y/N?   Value: 0x??
Virtual Memory

Q) What is the value of the address?
1. Extract more information

CO: 0x00   CI: 0x01   CT: 0x??   Cache Hit: Y/N?   Value: 0x??
Q) **What is the value of the address?**
1. Extract more information
2. Go to Cache Table

CO: 0x00  CI: 0x01  CT: 0x7F  Cache Hit: Y/N?  Value: 0x??
Q) **What is the value of the address?**

1. Extract more information
2. Go to Cache Table

CO: 0x00  CI: 0x01  CT: 0x7F  Cache Hit: Y  Value: 0x??
Virtual Memory

Q) What is the value of the address?
1. Extract more information
2. Go to Cache Table

CO: 0x00  CI: 0x01  CT: 0x7F  Cache Hit: Y  Value: 0xFF
Virtual Memory
Good luck!
Malloc

YOU
MALLOC
ALLOCATING ON THE STACK
Malloc

- Fit algorithms - first/next/best/good
- Fragmentation
  - Internal - inside blocks
  - External - between blocks
- Organization
  - Implicit
  - Explicit
  - Segregated
Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

```
a = malloc(32)
b = malloc(16)
c = malloc(16)
d = malloc(40)
  free(c)
  free(a)
e = malloc(16)
  free(d)
f = malloc(48)
  free(b)
```
Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>malloc(40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

free(a)
free(c)
free(d)
free(e)
free(b)
## Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>c = malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>d = malloc(40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>e = malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>f = malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Malloc - First fit

- **16 byte align**
- **coalesced**
- **footerless**
- **32 min size**

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#1</td>
<td>#2</td>
<td>#3</td>
<td>#4</td>
<td>#5</td>
<td>#6</td>
</tr>
<tr>
<td>---</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a 32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a 32a 32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a 32a 32a 48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a 32a 32f [0] 48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0] 32a 32f [1] 48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 16 byte align
- coalesced
- footerless
- 32 min size
### Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td></td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td>80f [0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Problem 10 Part A
# Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td>48a</td>
<td>32a</td>
<td>80f [0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td>48a</td>
<td>32a</td>
<td>80a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*final-f01.pdf: Problem 10 Part A*
Mallocc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td>80f [0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td>80a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td>48a</td>
<td>32f [0]</td>
<td></td>
<td>80a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?  
- internal?

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(d)</td>
<td>48a</td>
<td>32a</td>
<td>80f [0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>malloc(48)</td>
<td>48a</td>
<td>32a</td>
<td>80a</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(b)</td>
<td>48a</td>
<td>32f [0]</td>
<td>80a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?
  - internal
  - \((48-16) + (80-48) = 64\)
  - external?

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td>48a</td>
<td>32a</td>
<td>80f [0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td>48a</td>
<td>32a</td>
<td>80a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td>48a</td>
<td>32f [0]</td>
<td>80a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?
  - internal
  - (48-16) + (80-48) = 64
  - external
  - 32

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td>48a</td>
<td>32a</td>
<td>80f [0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td>48a</td>
<td>32a</td>
<td>80a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td>48a</td>
<td>32f [0]</td>
<td>80a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>malloc(32)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>malloc(40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{free}(c)\]
\[\text{free}(a)\]
\[\text{free}(d)\]
\[\text{free}(b)\]
## Malloc - Best fit

- **16 byte align**
- **coalesced**
- **footerless**
- **32 min size**

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f[0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f[0]</td>
<td>32a</td>
<td>32f[1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Maloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td>48f [1]</td>
<td>32a</td>
<td>32a</td>
<td>48f [0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Malloc - Best fit

- **16 byte align**
- **coalesced**
- **footerless**
- **32 min size**

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f[0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f[0]</td>
<td>32a</td>
<td>32f[1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48f[0]</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td>48f[1]</td>
<td>32a</td>
<td>32a</td>
<td>48f[0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td>48f[0]</td>
<td>32a</td>
<td>32a</td>
<td>54a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**final-f01.pdf**: Problem 10 Part A
### Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td>48f [1]</td>
<td>32a</td>
<td>32a</td>
<td>48f [0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32a</td>
<td>54a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td>48f [1]</td>
<td>32f [0]</td>
<td>32a</td>
<td>54a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?
- internal?

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>malloc(16)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(d)</td>
<td>48f [1]</td>
<td>32a</td>
<td>32a</td>
<td>48f [0]</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>malloc(48)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32a</td>
<td>54a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>free(b)</td>
<td>48f [1]</td>
<td>32f [0]</td>
<td>32a</td>
<td>54a</td>
<td></td>
</tr>
</tbody>
</table>
### Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?  
  - internal
  - (32-16) + (54-48) = 24
  - external?

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>malloc(16)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>malloc(40)</td>
<td>48a</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a</td>
<td>32a</td>
<td>32f [0]</td>
<td></td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32f [1]</td>
<td>48a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>malloc(16)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32a</td>
<td>48a</td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td>48f [1]</td>
<td>32a</td>
<td>32a</td>
<td>48f [0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>malloc(48)</td>
<td>48f [0]</td>
<td>32a</td>
<td>32a</td>
<td>54a</td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td>48f [1]</td>
<td>32f [0]</td>
<td>32a</td>
<td>54a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?
  - internal
  - (32-16) + (54-48) = 24
  - external
  - 48+32 = 80

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = malloc(32)</td>
<td>48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = malloc(16)</td>
<td>48a 32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c = malloc(16)</td>
<td>48a 32a 32a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d = malloc(40)</td>
<td>48a 32a 32a 48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(c)</td>
<td>48a 32a 32f [0] 48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(a)</td>
<td>48f [0] 32a 32f [1] 48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e = malloc(16)</td>
<td>48f [0] 32a 32a 48a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(d)</td>
<td>48f [1] 32a 32a 48f [0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = malloc(48)</td>
<td>48f [0] 32a 32a 54a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free(b)</td>
<td>48f [1] 32f [0] 32a 54a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Good luck!