Example Thread Problems

pthread_create - create a new thread

```
int pthread_create(pthread_t * thread, pthread_attr_t * attr, void * (*start_routi
*), void * arg);
```

pthread_self - return identifier of current thread

```
pthread_t pthread_self(void);
```

pthread_exit - terminate the calling thread

```
void pthread_exit(void *retval);
```

pthread_join - wait for termination of another thread

```
int pthread_join(pthread_t th, void **thread_return);
```

pthread_detach - put a running thread in the detached state

```
int pthread_detach(pthread_t th);
```

pthread_cancel - thread cancellation

```
int pthread_cancel(pthread_t thread);
```

Semaphores

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t * sem); /* P */
int sem_post(sem_t * sem); /* V */
```

Problem 1:

Consider the following programs which uses threads. Assume that all system calls return normally. Circle the programs which are guaranteed to print out 42.

Program 1

```
void *thread(void *vargp)
{
    pthread_exit((void*)42);
}

int main()
{
    int i;
    pthread_t tid;
    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, (void **)&i);

    printf("%d\n",i);
}
```

Program 2

```
void *thread(void *vargp)
{
    exit(42);
}

int main()
{
    int i;
    pthread_t tid;
    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, (void **)&i);

    printf("%d\n",i);
}
```

Program 3

```
void *thread(void *vargp)
{
    int *ptr = (int*)vargp;
    pthread_exit((void*)*ptr);
}

void *thread2(void *vargp)
{
    int *ptr = (int*)vargp;
    *ptr = 0;
    pthread_exit((void*)31);
}

int main()
{
    int i = 42;
    pthread_t tid, tid2;
    pthread_create(&tid, NULL, thread, (void*)&i);
    pthread_create(&tid2, NULL, thread2, (void*)&i);
    pthread_join(tid, (void**)&i);
    pthread_join(tid2, NULL);

    printf("%d\n",i);
}
```

Program 4

```
void *thread(void *vargp)
{
    pthread_detach(pthread_self());
    pthread_exit((void*)42);
}

int main()
{
    int i = 0;
    pthread_t tid;
    pthread_create(&tid, NULL, thread, (void*)&i);
    pthread_join(tid, (void**)&i);

    printf("%d\n",i);
}
```

Program 5

```
int i = 42;

void *thread(void *vargp)
{
    printf("%d\n",i);
}

void *thread2(void *vargp)
{
    i = 31;
}

int main()
{
    pthread_t tid, tid2;
    pthread_create(&tid2, NULL, thread2, (void*)&i);
    pthread_create(&tid, NULL, thread, (void*)&i);
    pthread_join(tid, (void**)&i);
    pthread_join(tid2, NULL);
}
```

Problem 2:

This problem tests your understanding of race conditions in concurrent programs.

Consider a simple concurrent program with the following specification: The main thread creates two peer threads, passing each peer thread a unique integer *thread ID* (either 0 or 1), and then waits for each thread to terminate. Each peer thread prints its thread ID and then terminates.

Each of the following programs attempts to implement this specification. However, some are incorrect because they contain a race on the value of `myid` that makes it possible for one or more peer threads to print an incorrect thread ID. Except for the race, each program is otherwise correct.

You are to indicate whether or not each of the following programs contains such a race on the value of `myid`.

A. **Does the following program contain a race on the value of `myid`?** Yes No

```
void *foo(void *vargp) {
    int myid;
    myid = *((int *)vargp);
    Free(vargp);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i, *ptr;

    for (i = 0; i < 2; i++) {
        ptr = Malloc(sizeof(int));
        *ptr = i;
        Pthread_create(&tid[i], 0, foo, ptr);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

B. **Does the following program contain a race on the value of `myid`?** Yes No

```
void *foo(void *vargp) {
    int id;
    id = *((int *)vargp);
    printf("Thread %d\n", id);
}

int main() {
    pthread_t tid[2];
    int i;

    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], NULL, foo, &i);
    Pthread_join(tid[0], NULL);
    Pthread_join(tid[1], NULL);
}
```

C. **Does the following program contain a race on the value of `myid`?** Yes No

```
void *foo(void *vargp) {
    int id;
    id = (int)vargp;
    printf("Thread %d\n", id);
}

int main() {
    pthread_t tid[2];
    int i;

    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], 0, foo, i);
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

D. **Does the following program contain a race on the value of `myid`?** Yes No

```
sem_t s; /* semaphore s */

void *foo(void *vargp) {
    int id;
    id = *((int *)vargp);
    V(&s);
    printf("Thread %d\n", id);
}

int main() {
    pthread_t tid[2];
    int i;

    sem_init(&s, 0, 0); /* S=0 INITIALLY */

    for (i = 0; i < 2; i++) {
        Pthread_create(&tid[i], 0, foo, &i);
        P(&s);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

E. **Does the following program contain a race on the value of `myid`?** Yes No

```
sem_t s; /* semaphore s */

void *foo(void *vargp) {
    int id;
    P(&s);
    id = *((int *)vargp);
    V(&s);
    printf("Thread %d\n", id);
}

int main() {
    pthread_t tid[2];
    int i;

    sem_init(&s, 0, 1); /* S=1 INITIALLY */

    for (i = 0; i < 2; i++) {
        Pthread_create(&tid[i], 0, foo, &i);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```