Recitation 14: Exam Review - Signals

Instructor: TA(s)

Outline

- Proxylab
- Final Exam
- TA Applications
- Signals

Proxylab

- **Proxylab is due Thursday (or late by Friday)**
  - **No submissions will be accepted after Friday!**
  - Submit something, even if doesn't pass everything
- **Worth almost a letter grade**
- **Submit early**
  - Autolab may compile / run differently if you have undefined behavior or race conditions

Final Exam Logistics

- Online but in-person

- 213/513 Fall 2021 Final Exam Review
  - Time: 12:00 pm - 3:00 pm on December 4th
  - Location: TBD

- Monitor Piazza for more information about the final exam

So you wanna TA for 213?

- **What qualifications are we looking for?**
  - **Decent class performance, but also critical thinking skills**
  - **Like computer systems + want to help others like systems!**
  - **Have a reasonable ability to gauge your schedule + responsibilities**
  - **Leadership potential! Take initiative, we love to see it** 😌
  - **Ability to tell students:**
    - **“Did you write your heap checker?”**
    - **“Run backtrace for me”**

Apply at https://www.ugrad.cs.cmu.edu/ta/S21/

Signals and Handling Reminders

- **Signals can happen at any time**
  - Control when through blocking signals
- **Signals also communicate that events have occurred**
  - What event(s) correspond to each signal?
- **Write separate routines for receiving (i.e., signals)**
  - What can you do / not do in a signal handler?

Signal Blocking

- We need to block and unblock signals. Which sequence?

```
pid_t pid;    sigset_t mysigs, prev;
sigemptyset(&mysigs);
sigaddset(&mysigs, SIGCHLD);
sigaddset(&mysigs, SIGINT);
// need to block signals. what to use?
// A. sigprocmask(SIG_BLOCK, &mysigs, &prev);
// B. sigprocmask(SIG_SETMASK, &mysigs, &prev);


if ((pid = fork()) == 0) {
    // need to unblock signals. what to use?
    /* A. sigprocmask(SIG_BLOCK, &mysigs, &prev);
     * B. sigprocmask(SIG_UNBLOCK, &mysigs, &prev);
     * C. sigprocmask(SIG_SETMASK, &prev, NULL);
     * D. sigprocmask(SIG_BLOCK, &prev, NULL);
     * E. sigprocmask(SIG_SETMASK, &mysigs, &prev);
```

Signal Blocking

- We need to block and unblock signals. Which sequence?

```
pid_t pid;    sigset_t mysigs, prev;
sigemptyset(&mysigs);
sigaddset(&mysigs, SIGCHLD);
sigaddset(&mysigs, SIGINT);
// need to block signals. what to use?
// A. sigprocmask(SIG_BLOCK, &mysigs, &prev);
// B. sigprocmask(SIG_SETMASK, &mysigs, &prev);


if ((pid = fork()) == 0) {
    // need to unblock signals. what to use?
    /* A. sigprocmask(SIG_BLOCK, &mysigs, &prev);
     * B. sigprocmask(SIG_UNBLOCK, &mysigs, &prev);
     * C. sigprocmask(SIG_SETMASK, &prev, NULL);
     * D. sigprocmask(SIG_BLOCK, &prev, NULL);
     * E. sigprocmask(SIG_SETMASK, &mysigs, &prev);
```

Signal Blocking

- We need to block and unblock signals. Which sequence?

```
pid_t pid;    sigset_t mysigs, prev;
sigemptyset(&mysigs);
sigaddset(&mysigs, SIGCHLD);
sigaddset(&mysigs, SIGINT);
// need to block signals. what to use?
// A. sigprocmask(SIG_BLOCK, &mysigs, &prev);
// B. sigprocmask(SIG_SETMASK, &mysigs, &prev);


if ((pid = fork()) == 0) {
    // need to unblock signals. what to use?
    /* A. sigprocmask(SIG_BLOCK, &mysigs, &prev);
     * B. sigprocmask(SIG_UNBLOCK, &mysigs, &prev);
     * C. sigprocmask(SIG_SETMASK, &prev, NULL);
     * D. sigprocmask(SIG_BLOCK, &prev, NULL);
     * E. sigprocmask(SIG_SETMASK, &mysigs, &prev);
```

Signal Blocking cont.

- Someone implemented the wrong choices. Which signals are now blocked?

```
pid_t pid;    sigset_t mysigs, prev;
sigemptyset(&mysigs);
sigaddset(&mysigs, SIGCHLD);
sigaddset(&mysigs, SIGINT);


sigprocmask(SIG_SETMASK, &mysigs, &prev);
// What is blocked?


if ((pid = fork()) == 0) {
  sigprocmask(SIG_BLOCK, &prev, NULL);
  // What is blocked?
```

Signal Queuing

- How many times is the handler invoked?

```
void handler(int sig)
{ ...}


...
sigset_t mysigs, prev;
signal(SIGUSR1, handler);
sigemptyset(&mysigs);
sigaddset(&mysigs, SIGUSR1);
sigprocmask(SIG_BLOCK, &mysigs, &prev);
kill(getpid(), SIGUSR1);
kill(getpid(), SIGUSR1);
sigprocmask(SIG_SETMASK, &prev, NULL);
```

Signal Delivery

- What can be printed?
- When is a blocked signal delivered?

```
sigset_t mysigs, prev;
sigemptyset(&mysigs);
sigaddset(&mysigs, SIGINT);
sigprocmask(SIG_BLOCK, &mysigs, &prev);
pid_t pid = fork();


if (pid > 0) {
    kill(pid, SIGINT);
    sigprocmask(SIG_SETMASK, &prev, NULL);
    printf("A");
} else {
    kill(getppid(),SIGINT);
    sigprocmask(SIG_SETMASK, &prev, NULL);
    printf("B");
}
```

Signal Delivery

- Child calls kill(parent, SIGUSR{1,2}) between 2-4 times.
  What sequence of kills may print 1?
  Can you guarantee printing 2?
  What is the range of values printed?

```
int counter = 0;
void handler (int sig) {
  atomically {counter++;}
}
int main(int argc, char** argv) {
  signal(SIGUSR1, handler);
  signal(SIGUSR2, handler);
  int parent = getpid();   int child = fork();
  if (child == 0) {
    /* insert code here */
    exit(0);
  }
  sleep(1);   waitpid(child, NULL, 0);
  printf("Received %d USR{1,2} signals\n", counter);
}
```


See http://www.cs.cmu.edu/~213/oldexams/final-s07.pdf

Signal Delivery

- Suppose the program is currently inside the signal handler, which signals are blocked?
- Is this handler safe?

```
int counter = 0;
void handler (int sig)
{
  counter++;
}
int main(int argc, char** argv)
{
  signal(SIGUSR1, handler);
  signal(SIGUSR2, handler);
}
```

Final Exam Q&A

Appendix: Thread Synchronization

You can assume `pthread_create` and `pthread_join` executed successfully. And `printf` always flushes stdout.

```
sem_t add_sem;
sem_t rem_sem;

void add() {
    printf("A");
}

void remove() {
    printf("R");
}

void *thread1(void *vargp) {
    V(&add_sem);
    V(&rem_sem);

    remove();

    P(&add_sem);
    P(&rem_sem);

    add();

    V(&add_sem);
    V(&rem_sem);

    remove();
    add();
}

void *thread2(void *vargp) {
    P(&rem_sem);
    P(&add_sem);

    add();
    remove();
}
```

```
int main() {
    pthread_t tid1, tid2;

    sem_init(&add_sem,0,0);
    sem_init(&rem_sem,0,0);

    pthread_create(&tid1, NULL, thread1, NULL);
    pthread_create(&tid2, NULL, thread2, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

    return 0;
}
```

1. How many potential deadlock situations are present in the above code?
2. For lengths 0-6, list the number of possible outcomes of that length that can be produced.

Appendix: Thread Synchronization (Contd.)

Now, we redefine the thread1 and thread2 functions and add a global variable i, but keep main the same. (Main is still shown for easy reference.)

```
int i = 0;
sem_t add_sem;

void *thread1(void *vargp) {
    V(&add_sem);
    for (i = 0; i < 2; i++);
}

void *thread2(void *vargp) {
    for (int count = 0; count < 2; count++){
        P(&add_sem);
        printf("%d", i);
        V(&add_sem);
    }
}

int main() {
    pthread_t tid1, tid2;

    sem_init(&add_sem,0,0);

    pthread_create(&tid1, NULL, thread1, NULL);
    pthread_create(&tid2, NULL, thread2, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

    return 0;
}
```

How many outcomes are possible?