Today

- Signals (in depth)
- Threads
- Midterm 2 Review
Signals Recap

- Each signal is identified by a number.
- When a signal is received the following can happen:
  1. default action
  2. signal is ignored or blocked
  3. signal is caught and handled

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Signal Number</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGINT</td>
<td>2</td>
<td>CTRL + C</td>
</tr>
<tr>
<td>SIGALRM</td>
<td>14</td>
<td>Timer signal from alarm(2)</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>11</td>
<td>Invalid memory reference</td>
</tr>
</tbody>
</table>
Signals API (1)

- #include <signal.h>
- Set signal handler (older API)
  - typedef void (*sighandler_t)(int);
  - sighandler_t signal(int signum, sighandler_t handler);
- Change the action of a process for a signal (recommended)
  - int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
- Send signals
  - int kill(pid_t pid, int sig);
- Send alarm signal
  - unsigned int alarm(unsigned int seconds);
- Block signals
  - int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
signal()

- Ignore SIGINT(2)
  - `signal(SIGALRM, SIG_IGN);`
- Catch SIGINT(2)
  - Define a handler function
    ```c
    void my_handler(int signum) {
        printf("OUCH! - I got signal %d\n", signum);
    }
    ```
  - Set user-defined signal handler
    `signal(SIGINT, my_handler);`
- Restore default action
  - `signal(SIGINT, SIG_DFL);`
- Return errors?
  - Check the man page by yourself
`sigaction()`

```c
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

- The `sigaction` structure has at least the following members:
  - void (*) (int) sa_handler /* function, SIG_DFL or SIG_IGN
  - sigset_t sa_mask /* signals to block in sa_handler
  - int sa_flags /* signal action modifiers

- Example:
  ```c
  struct sigaction new_action;
  new_action.sa_handler = my_handler;
  new_action.sa_flags = 0; //special options
  sigemptyset(&new_action.sa_mask);
  //signals to block – How to do it?
  sigaction(SIGINT, &new_action, NULL);
  ```
sigprocmask()

- Each process has a signal mask, which indicates which signals are blocked.
- The signal mask can be fetched or changed with:
  ```c
  int sigprocmask(int how, sigset_t *set, sigset_t *oldset);
  ```
- `how` can be one of:
  - `SIG_BLOCK`: add signals in `set` to the signal mask
  - `SIG_UNBLOCK`: remove signal in `set` from the signal mask
  - `SIG_SETMASK`: the signal mask is set from `set`
- Manipulation of signal set
  ```c
  int sigaddset(sigset_t *set, int signo);
  int sigemptyset(sigset_t *set);
  int sigfillset(sigset_t *set);
  int sigdelset(sigset_t *set, int signo);
  ```
kill(), raise()

**kill(pid_t pid, int sig);**

- If: \( pid > 0 \) : send to process with that pid
- \( pid = 0 \) : sends signal to members of caller’s process group
- \( pid = -1 \) : sends signals to all process which it has permission

**raise(int sig);**

The raise() function sends a signal to the calling process or thread. In a single-threaded program it is equivalent to kill(getpid(), sig);
SIGALRM, alarm(), pause()

- SIGALRM is a signal sent to a process once a time limit has elapsed.
- Can be used to make a long-running action time out or to perform an action periodically.
- Can be scheduled from `alarm(sec)`.
- SIGALRM and alarm are used to implement `sleep()`.
- `pause()` causes the program to suspend execution until a signal occurs.

```c
#include <unistd.h>
unsigned int alarm(unsigned int seconds);
int pause(void);
```
Exercise 1

1. Run ‘time ./Signal_correct’
   [=====]
   0.000u 0.000s 0:10.00 0.0% 0+0k 32+0io 0pf+0w

2. Compile and run Signal.c
   [====]
   =0.000u 0.000s 0:10.00 0.0% 0+0k 0+0io 0pf+0w

3. Fix Signal.c, so that the last ‘=’ appears on the same line.
   [Hint] change 1 line of code
• Creating a thread is like a combination of `fork()` and `exec()`
• `#include <pthread.h>`
  ```c
  int pthread_create(
      pthread_t *thread,
      pthread_attr_t *attr,
      void (*function)(void *),
      void *arg);
  ```
• `thread` is the returned thread ID, `attr` is an attribute set, `function` is the function to be called with `arg`
• Compile/Link with `‘-pthread’` option
pthread_join(), pthread_exit()

- int pthread_join(pthread_t thread, void **retval);
  - wait until a thread exits
- void pthread_exit(void *retval);
  - exit from current thread
Exercise 2

Bug fix

1. Run ‘hello_arg1’
2. Compile & run hello_arg1.c, what’s the issue?
3. Fix hello_arg1.c
   [Hint] There are two ways to fix the issue
Exercise 3

Learn how to pass multiple parameters to thread

1. Run ‘hello_arg2’
2. Complete hello_arg2.c
Exam Review

• Questions from you?

• Some review of topics thus far
  – IPC (Pipes, Shared Memory, Message Queue)
  – Signals
  – Threads
Pipes

- Processes can create a pipe internally:
  - `void pipe(int fd[2])`
- Takes an array of two `ints`, creates a pipe, and places the input and output file descriptors in the array.
  - `fd[0]` is the “read” end of the pipe,
  - `fd[1]` is the “write” end.
Pipes – <Non blocking I/O>

• Non-blocking I/O is possible by setting O_NONBLOCK flag.
• Use fcntl() system call to set any flag for Input/Output file descriptor.
  – fcntl(fd[0], F_SETFL, O_NONBLOCK)
Shared Memory

- Creating the segment and connecting
  - int shmget(key_t key, size_t size, int shmflg)
  - shmid = shmget(key, 1024, 0644 | IPC_CREAT);

- Attach – getting a pointer to the segment
  - void* shmat(int shmid, void *shmaddr, int shmflg)
  - data = shmat(shmid, (void *) 0, 0);
Shared Memory

- **Reading and writing**
  - Returned data is a string, so you can use `printf`
  - `printf("shared contents: %s\n", data);`

- **Detaching from and deleting segments**
  - **Detach** – `int shmdt(void *shmaddr)`
  - **Destroy** – `shmctl(shmid, IPC_RMID, NULL)`
Message Queue

- Create/connect to a message queue
  - `int msgget(key_t key, int msgflg)`
  - `key` is a system-wide unique identifier
  - `msgflg` tells `msgget()` what to do with the queue
  - `msgid = msgget(key, 0666 | IPC_CREAT)`
Message Queue

- Sending to the queue
  - struct msgbuf {
    long mtype;
    char mtext[100];
  };
  - int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
  - msqid is the id of the queue
  - msgp is a pointer to the message
  - msgsz is the actual size of the message
  - msgflg indicates other options, you can set it to 0
Message Queue

• Receiving from the queue
  - int msgrcv(int msqid, void *msgp, size_t msgsiz, long msgtyp, int msgflg)
  - msgp is the buffer to store the message
  - msgsiz is the size of the message
  - msgtyp is used to identify which message to receive
  - msgflg indicates other options, you can set it to 0
Message Queue

- Destroying a message queue
  - `#include <sys/msg.h>`

    `msgctl(msqid, IPC_RMID, NULL);`

You can destroy the IPC by `ipcrm` command.