Operating Systems: Heavyweight and Lightweight Concurrency in UNIX

- Today we'll cover:
  - fork/exec
  - sockets
  - signals
  - pthreads

Tips

- Ignore or note any details you don't understand.
  - The slides will be available online.
- Ignore details of any UNIX command or function reference you don't know.
  - The UNIX “man” command is your friend.
  - Example: “man man” or “man 2 <*syscall*>”
- We've covered a lot of the theory already. Focus on strengthening connections between course concepts and the UNIX library functions.

Heavyweight Concurrency in UNIX: processes

- Process creation:
  - fork()/exec()
- Process synchronization:
  - wait()
- Inter-Process Communication:
  - A whole bunch of stuff we'll breeze through.
- Each process runs in its own memory space.

Process Creation: fork()

- Prototype:
  - pid_t fork(void);
- Inputs:
  - None
- Outputs:
  - Another process!
  - Child's return value is 0.
  - Parent's return value is the child's process ID.

Spawning Other Code: exec()

- Prototype (1 of 5):
  - int execv(const char *path,
    char *const argv[]);
- Inputs:
  - Full path of program file name.
  - Command line argument variables.
- Outputs:
  - Replaces current process with code from other executable.
  - If you get a return code, something went wrong.

Process Synchronization: wait()

- Prototype:
  - pid_t wait(int * status);
- Inputs:
  - Address of output variable.
- Outputs:
  - Process ID of terminated child.
  - Return code of terminated child.
- Other forms:
  - pid_t waitpid(pid_t pid, int * status, int options);

Inter-Process Communication

- Pipes
- Sockets
- Signals
- *shared memory*
  - *shmat(), shmctl(), shmget(), mmap()*
- *semaphores*
  - *POSIX: sem_init(), sem_*()*
  - *System V: semctl(), semop()*

IPC: Pipes

- Pipe creation.
  - pipe(), dup()
- Pipe communication.
  - read(), write()
- Pipe destruction.
  - int close(int fd);
- Pipe example.

Pipe creation: pipe()

- Prototype:
  - int pipe(int filedes[2]);
- Inputs:
  - Address of a pair of memory cells for output.
- Outputs:
  - Result code.
  - Pair of file descriptors (output via parameter).
    - Read from filedes[0], write to filedes[1].

Pipe creation: dup() (and dup2())

- Prototype:
  - int dup(int oldfd);
  - int dup2(int oldfd, int newfd);
- Inputs:
  - File descriptor to duplicate.
  - dup2(): File descriptor to replace.
- Outputs:
  - File descriptor of duplicate.

Pipe example

```
void pipe_demo (void)
{
  int fds [2];
  pid_t child_pid = -1;

  pipe(fds);
  child_pid = fork();
  if (child_pid == 0)
    {
      close(fds[0]);
      dup2(fds[1], 1); /* Make stdout go to the pipe. */
      printf("%d\n", 42);
    }
  else
    {
      int child_result;
      close(fds[1]);
      dup2(fds[0], 0); /* Make stdin come from the pipe. */
      scanf("%d\n", &child_result);
      printf("Got %d from child.\n", child_result);
      waitpid(child_pid, NULL, 0);
    }
}
```

IPC: Sockets

- Socket creation.
  - Client socket creation.
  - Server socket creation.
- Socket communication.
- Socket destruction.
  - int close(int fd);
- Socket example.

Socket creation: client

- Create the socket using socket().
- Connect to the server using connect().

Socket creation: socket()

- Prototype:
  - int socket(int domain, int type, int protocol);
- Inputs:
  - Networking domain (AF_INET, AF_UNIX)
  - Connection type (SOCK_STREAM, SOCK_DGRAM)
  - Protocol: Unused for common domains.
- Outputs:
  - File descriptor (or <0 on error).

Socket creation: connect()

- Prototype:
  - int connect(int sockfd,
    const struct sockaddr *serv_addr,
    socklen_t addrlen);
- Inputs:
  - Socket file descriptor.
  - Address data structure and data structure size.
- Output:
  - Success code (<0 on failure).

Example: create_client_socket()

```
int create_client_socket (const char * hostname, int hostport)
{
  int ret_val = -1;
  struct sockaddr_in servaddr;

  ret_val = socket(AF_INET, SOCK_STREAM, 0);
  if (ret_val >= 0) {
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(hostport);
    if (inet_pton(AF_INET, hostname, &servaddr.sin_addr) <= 0) {
      close(ret_val);
      ret_val = -2; /* Address conversion failed. */
    } else {
      if (connect(ret_val, (struct sockaddr *) &servaddr,
                  sizeof(servaddr)) != 0) {
        close(ret_val);
        ret_val = -3; /* Connection failed. */
      }
    }
  }
  return ret_val;
}
```

Socket creation: server

- Create the socket: socket()
- Bind the socket to an address: bind()
- Inform the OS you are listening for connection: listen()
- Create sockets for incoming connections: accept()

Example: create_server_socket()

```
int create_server_socket (const char * addr, int port)
{
  int ret_val = -1;
  struct sockaddr_in listenaddr;

  ret_val = socket(AF_INET, SOCK_STREAM, 0);
  if (ret_val >= 0) {
    memset(&listenaddr, 0, sizeof(listenaddr));
    listenaddr.sin_family = AF_INET;
    listenaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    listenaddr.sin_port = htons(port);
    if (bind(ret_val, (struct sockaddr *) &listenaddr,
             sizeof(listenaddr)) != 0) {
      close(ret_val);
      ret_val = -2; /* bind() failed. */
    } else {
      if (listen(ret_val, 1) != 0) {
        close(ret_val);
        ret_val = -3; /* listen() failed. */
      }
    }
  }
  return ret_val;
}
```

Example: wait_for_client_connection()

```
int wait_for_client_connection (int listenfd)
{
  int ret_val = -1;
  socklen_t clilen;
  struct sockaddr_in cliaddr;

  memset(&cliaddr, 0, sizeof(cliaddr));
  clilen = sizeof(cliaddr);
  ret_val = accept(listenfd, (struct sockaddr *) &cliaddr,
                   &clilen);
  return ret_val;
}
```

Socket communication

- Can use file descriptor I/O
  - read() and write()
  - Similar to pipes at this point.
- Additional functions specific to sockets:
  - ssize_t recv(int s, void *buf, size_t len, int flags);
  - ssize_t send(int s, const void *buf, size_t len, int flags);

Socket example

```
void socket_demo (int port)
{
  pid_t child_pid = fork();
  if (child_pid == 0) {
    int toserver_fd = -1;
    sleep(1); /* Here we have our first race condition! */
    toserver_fd = create_client_socket("127.0.0.1", port);
    if (toserver_fd >= 0) {
      write(toserver_fd, "Hi there!\0", 10);
      close(toserver_fd);
    }
  } else {
    char buffer [80];
    int server_fd = create_server_socket(port);
    if (server_fd >= 0) {
      int fromcli_fd = wait_for_client_connection(server_fd);
      if (fromcli_fd >= 0) {
        read(fromcli_fd, buffer, 80);
        close(fromcli_fd);
        printf("Got '%s' from the client.\n", buffer);
      }
      waitpid(child_pid, NULL, 0);
      close(server_fd);
    }
  }
}
```

IPC: Signals

- Sending a signal.
  - kill()
- Handling a signal.
  - signal() - Sets a signal handler function.

Example: signal_demo()

```
void sigint_handler (int sig) {
  printf("Parent(%d): Masking SIGINT...\n", getpid());
}

void child_sigint_handler (int sig) {
  printf("Child(%d): Ahhh...got me!\n", getpid());
  exit(0);
}

void signal_demo (void)
{
  pid_t child_pid = fork();
  if (child_pid == 0) {
    signal(SIGINT, child_sigint_handler);
    while (1) {
      sleep(100);
    }
  } else {
    signal(SIGINT, sigint_handler);
    printf("Parent(%d): Waiting for Ctrl-C...\n", getpid());
    pause();
    kill(child_pid, SIGINT);
    waitpid(child_pid, NULL, 0); /* Verify join... */
  }
}
```

“Lightweight” Concurrency in UNIX: threads

- Thread creation/destruction:
  - pthread_create(), pthread_exit()
- Thread synchronization:
  - *pthread_join()*
  - Mutexes
  - Condition variables
- Thread communication and other issues.
- Each thread runs in the same memory space (with different stacks).

Thread creation: pthread_create()

- Prototype:
  - int pthread_create(
    pthread_t * thread, pthread_attr_t * attr,
    void * (*start_routine)(void *), void * arg);
- Inputs:
  - Memory to put the thread ID.
  - Thread attributes.
  - Function to run in new thread.
- Outputs:
  - Thread ID, result code (0 on success).

Thread destruction: pthread_exit()

- Prototype:
  - void pthread_exit(void *retval);
- Inputs:
  - Value to return to joining thread.
- Outputs:
  - Does not return.

Threading Example

```
void * writer (void * arg) {
  while (1) {
    printf("%d: I would be sending a greeting now.\n",
           (int)pthread_self());
    sleep(1); /* One hopes this doesn't use SIGALRM */
  }
  return NULL;
}

void * reader (void * arg) {
  while (1) {
    printf("%d: I would be reading something now.\n",
           (int)pthread_self());
    sleep(1);
  }
  return NULL;
}

void pthread_demo (void)
{
  pthread_t tid_r, tid_w;
  void * dummy;
  pthread_create(&tid_r, NULL, reader, NULL);
  pthread_create(&tid_w, NULL, writer, NULL);
  pthread_join(tid_r, &dummy);
  pthread_join(tid_w, &dummy);
}
```

Thread Synchronization: Mutexes

- Mutex construction/destruction:
  - pthread_mutex_init(), pthread_mutex_destroy()
- Acquiring the mutex lock:
  - pthread_mutex_lock()
  - pthread_mutex_trylock()
- Releasing the mutex lock:
  - pthread_mutex_unlock()

Mutex Example

```
static pthread_mutex_t buf_lock = PTHREAD_MUTEX_INITIALIZER;
static char * buf = NULL;

static void * writer (void * arg) {
  while (1) {
    pthread_mutex_lock(&buf_lock);
    if (buf == NULL) {
      printf("%d: Sending greeting.\n", (int)pthread_self());
      buf = strdup("Hi there!");
    }
    pthread_mutex_unlock(&buf_lock);
    sleep(1); /* One hopes this doesn't use SIGALRM */
  }
  return NULL;
}

static void * reader (void * arg) {
  while (1) {
    pthread_mutex_lock(&buf_lock);
    if (buf != NULL) {
      printf("%d: Got '%s'\n", (int)pthread_self(), buf);
      free(buf);
      buf = NULL;
    }
    pthread_mutex_unlock(&buf_lock);
    sleep(1);
  }
  return NULL;
}
```

Thread Synchronization: Condition Variables

- Creation/destruction:
  - pthread_cond_init(), pthread_cond_destroy()
- Signaling
  - pthread_cond_signal(), pthread_cond_broadcast()
- Waiting
  - pthread_cond_wait(), pthread_cond_timedwait()
- No more sleeps!

Condition Variable Example

```
static pthread_mutex_t buf_lock = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t buf_cond_mt = PTHREAD_COND_INITIALIZER;
static pthread_cond_t buf_cond_full = PTHREAD_COND_INITIALIZER;
static char * buf = NULL;

static void * writer (void * arg) {
  while (1) {
    pthread_mutex_lock(&buf_lock);
    while (buf != NULL) {
      pthread_cond_wait(&buf_cond_mt, &buf_lock);
    }
    printf("%d: Sending message...\n", (int)pthread_self());
    buf = strdup("Hi there!");
    pthread_cond_signal(&buf_cond_full);
    pthread_mutex_unlock(&buf_lock);
  }
  return NULL;
}

static void * reader (void * arg) {
  while (1) {
    pthread_mutex_lock(&buf_lock);
    while (buf == NULL) {
      pthread_cond_wait(&buf_cond_full, &buf_lock);
    }
    printf("%d: Got '%s'\n", (int)pthread_self(), buf);
    free(buf); buf = NULL;
    pthread_cond_signal(&buf_cond_mt);
    pthread_mutex_unlock(&buf_lock);
```