15-213/18-243
Introduction to Computer Systems
Spring 2009
Recitation 7

Your Name Here

Overview

- System Calls
  - fork()
  - execve()
  - wait()
  - exit()
- Signals
- Shell Lab
  - Race Conditions
  - I/O Redirection

System Calls

- fork()
  - Splits the execution of one process into two processes which execute concurrently.
  - Returns the PID of the child process in the parent.
  - Returns 0 in the child.
  - The parent's address space is COPIED into the child's address space.
    - Writes to variables, arrays, etc. are not visible in the parent
    - File descriptors (stdin, stdout, stderr) are SHARED.

System Calls

- execve(char* path, char** argv, char** envp)
  - Loads the specified executable file and executes it in the current process.
  - argv is the argument vector passed to main().
  - envp is the environment variable list.
  - This is a common form of the exec system call – others exist (execv, execl, etc.).

System Calls

- exit(int status)
  - Causes the current process to terminate and sets the exit status.
  - Exit status 0 usually denotes success, any other value denotes failure.
  - Wait can retrieve the status set by exit().

System Calls

- wait(int* child_status)
  - Waits (blocks) until ANY child of the current process terminates.
  - Stores the exit status of the child in *child_status and then returns to the parent.

System Calls

- waitpid(int child_pid, int* child_status, int options)
  - Waits for a specific child to exit, or all children if child_pid = -1.
  - Blocking and other behavior can be configured via the options variable (a bit vector)
    - WNOHANG – don't block if no children have exited, just return 0
    - WUNTRACED – also return if the child has only stopped (received the SIGSTOP signal), not exited

Signals

- Overview
  - Signals are “sent” to a process when an exceptional event occurs.
  - Process “receives” signals by jumping to a signal handler function.
  - Signals can be received (and thus signal handlers may be called) *at any time during process execution*.

Signals

- **Useful Signals**
  - **SIGINT – sent when the interrupt sequence (usually Ctrl-C) is entered.**
  - **SIGSTOP – sent when the stop sequence (usually Ctrl-Z) is entered.**
  - **SIGCHLD – sent when any child process exits**
  - **SIGSEGV – sent when a process makes an improper request to read or write memory (SEGmentation Violation).**

Signals

- Signals are not queued!
  - If the same signal is sent more than once to a process before the process receives it, the signal handler function for the process will only be called once.
  - For example, if two children exit before the signal handler is called in the parent, the handler for SIGCHLD will only be called once.

Signals

- Blocking Signals
  - Signals can be blocked (rather, deferred) so that the program does not run a signal handler during a "critical section" of code.
  - Use the sigprocmask() API to block and unblock signals.
  - Signals sent to a process while blocked will be received when (if) the process unblocks them.
  - Again, if multiple signals of the same type are sent to a process while the signal is blocked, the signal will only be received ONCE when the signal is unblocked.

Shell Lab

- Overview
  - Your goal is to create a functioning Unix shell with foreground and background processes.
  - Exercises the process control system calls as well as signal handling.
  - You must write a main command line evaluation function which parses the command line and handles the creation of child processes.
  - You must also write signal handlers to detect and clean up terminated child processes.
  - Beware of race conditions!

Shell Lab

- Race Conditions
  - A race condition occurs when the state of a program is *unexpectedly nondeterministic*.
  - Nondeterminism occurs when multiple concurrent threads of execution operate upon a shared set of data, and the final value of the data changes depending on the order in which the threads are scheduled.
  - If a piece of code unexpectedly produces nondeterministic results, that code is said to have a race condition.

Shell Lab

- **Race Conditions and Signals**
  - **Recall that, in the absence of blocking, signals can be received by a program at any time.**
  - **Suppose both the main thread of execution and a signal handler modify shared state.**
  - **If the main thread is modifying this state and the signal handler is called during this operation, the signal handler sees *inconsistent state*.**
  - **If the signal handler then decides to use the shared state, the program may begin to behave in unexpected ways (if you are lucky, it will crash).**

Shell Lab

Race Conditions and Signals: A Trivial Example

```
int i = 0;

void sigalrm_handler() {
  i++;
}

int main(...) {
  int temp;
  /* set up signal handling */

  alarm(1);
  temp = i;
  temp = temp + 1;
  i = temp;
  pause();

  printf("%d", i);
  return 0;
}
```

What is the output?

Depends on when the signal handler is called!

If the alarm expires during the execution of these three lines, the program outputs 1 instead of 2!

Shell Lab

Race Conditions and Signals: A Trivial Example

```
int i = 0;

void sigalrm_handler() {
  i++;
}

int main(...) {
  int temp;
  /* set up signal handling */

  alarm(1);
  sigprocmask(SIG_BLOCK, SIGALRM);
  temp = i;
  temp = temp + 1;
  i = temp;
  sigprocmask(SIG_UNBLOCK, SIGALRM);
  pause();

  printf("%d", i);
  return 0;
}
```

How do we make this program deterministic (i.e. make it output 2 on every execution)?

Block the alarm signal during the critical section!

Critical Section

Shell Lab

- I/O Redirection
  - You may need to set the stdin or stdout of a child process to file descriptors referencing files on disk.
  - Use `dup2(old_fd, new_fd)` to do this.
  - For example, this redirects stdout to a file:

```
int main(int argc, char** argv) {
  int fd = open("/tmp/foo", O_TRUNC);
  dup2(fd, STDOUT_FILENO);
  printf("Hello World!");
  return 0;
}
```

    /tmp/foo will contain the text "Hello World!"

  - Be careful where you call dup2 in your shell!

Shell Lab

- There are lots more hints in the shell lab handout, be sure to read these!
- Be sure to review the lecture slides if you get stuck.
- Come to office hours or email the staff list if you have any questions.

Questions?