CSE 351 Section 8

Fork and Execve

Lab #4

- Any questions on Lab 4?
- This is a dummy cache - it's not affected by your in-memory arrays; those arrays go to your *real* cache
- Other questions in office hours

fork() basic functionality

- Copies the entire current process and runs it concurrently as a separate process
- Returns twice:
  - To the new child process: 0
  - To the parent process: the child's process ID (pid)
- Use return value to distinguish which process is currently being executed

fork() semantics

- What needs to be copied when fork() is called?
- Goal is for child to run identically
- All memory owned by the parent process must be copied to the child process
  - The address space is usually not copied all at once
  - **Copy-on-write**: OS copies a page of memory only when the child ***writes*** to the page
- Other things are copied, too, but copying memory is the most expensive operation

wait()

- Parent process waits for the child process to return (exit) before continuing
- Returns the process id (pid) of the child
- Takes an optional parameter
  - Type (int *), pointer to an int
  - Receives the exit status of the child process

waitpid()

- Just like wait(), but takes a pid as a parameter, so it is only waiting for a specific process to return
- Useful when you have many child processes and want finer control

Wait/Fork example code

```
#include <stdio.h>      /* printf, stderr, fprintf */
#include <sys/types.h>  /* pid_t */
#include <unistd.h>     /* _exit, fork */
#include <stdlib.h>     /* exit */
#include <errno.h>      /* errno */

int main(void)
{
   pid_t pid;
   pid = fork();
   if (pid == -1)
   {
      /* Fork didn't work */
      exit(-1);
   }
   if (pid == 0)
   {
      /* Child Process: do stuff here */
      _exit(return_status);
   }
   else
   {
      /* Parent Process: wait and then exit
      int return_status;
      waitpid(pid,&return_status,NULL);
      /* Do stuff once child has exited */
      exit(0);
   }
   return 0;
}
```

execve()

- Member of the exec() family of functions
- Executes a program in the current process
- Doesn't return unless there is an error
- Parameters:
  - Path to the executable
  - Array of arguments
  - Environment variables (tip on next slide)

Subtle exec() tip

- For commands without path ('ls' vs '/bin/ls')
- Some exec() calls try to load environment variables to resolve the path
- Add this declaration before main():
  - `extern char **environ`
- Now pass NULL as an environment variable argument to the exec() call

  `Ex: execve("/bin/pwd", NULL, NULL);`

Fork-Exec model

- Exec is powerful, but it is limited because it never returns
- What if you want to call another executable within your program?
  - Fork a child process
  - Use exec() in the child process
  - Make the parent wait for the child to exit
  - Continue working in the parent process

Fork-Exec example code

```
#include <stdio.h>      /* printf, stderr, fprintf */
#include <sys/types.h>  /* pid_t */
#include <unistd.h>     /* _exit, fork */
#include <stdlib.h>     /* exit */
#include <errno.h>      /* errno */

extern char **environ;

int main(void)
{
   pid_t pid;
   pid = fork();
   if (pid == -1)
   {
      /* Fork didn't work */
      exit(-1);
   }
   if (pid == 0)
   {
      /* Child process, call exec */
      execve("/bin/ls",NULL,NULL);
   }
   else
   {
      /* Parent Process: wait and then exit
      int return_status;
      waitpid(pid,&return_status,NULL);
      /* Do stuff once child has exited */
      exit(0);
   }
   return 0;
}
```

Sample programs

- Will be available on the class calendar later
- One is a simple fork/wait example
- One shows how the fork-exec model works

Do demos for the rest of class