Unix Process Management

Classic BSD Unix PCB
(http://src.gnu-darwin.org/src/sys/sys/proc.h.html)

Unix Processes: Creation and Termination

- **PID (process ID)** - unique number used to identify each process
- Process creation system calls
  - **fork**
  - **exec**
- Process destruction system call
  - **kill**

Unix fork system call

- Parent process (creator, old process)
- Child process (created, new process)
- After fork, two processes exist and are running concurrently.
- **fork** causes a new process to be created with a duplicate of the parent’s code
- fork returns a value of 0 to the child.
- fork returns the child’s PID to the parent
  - (greater than 0)
Interrupts

Unix fork system call

```c
/* spawn new process using fork */
main() {
  pid_t pid;
  printf("Just one process so far\n");
  printf("Calling fork\n");
  pid = fork(); /* create new process */
  if (pid == 0)
    printf("I am the child\n");
  else if (pid > 0)
    printf("I am the parent; my child has pid \%ld\n", pid);
  else
    printf("fork returned error code, no child\n");
}
```

Unix exec system call

- Replaces old process with new code.
- After exec, the process is running a new piece of code.
- Process ID is the same.
- exec never returns (unless there is an error).
- exec provides a means for passing parameters from the old to the new.
- Family of exec system calls differ in where the code comes from and how parameters are passed.

Unix exec system call

- **BEFORE exec**
- ```c
   printf("hello\n");
   Pid = execl("bin/ls");
   printf("never\n");
   ```
- **AFTER exec**
- ```c
   Code for ls
   ```
- Process A
- Process B (Note that Process A no longer exists.)

Unix process creation

```c
int main( void ) {
  pid_t pid; /* type is from sys/types.h */
  int error_code;
  pid = fork();
  if( pid == 0 ) {
    printf( "I am the child, running %s.\n", program_name );
    /* never returns if all goes well */
    error_code = execl( "/bin/program_name", program_name, "-l", (char*)0 );
    printf( "Error running ls, execl returned %d.\n",error_code );
    exit(0);
  }
  else {
    printf( "I am the parent process.\n");
    printf("My child is pid %ld\n", (long)pid);
    exit(0);
  }
}
```

Unix process creation (cont)

```c
else {
  printf( "I am the parent process.\n");
  printf("My child is pid %ld\n", (long)pid);
  exit(0);
  }
```

Unix process id system calls

- getpid - returns the pid of this process
- getppid - returns pid of this process’ parent
Unix process monitoring

- `ps` command
- `prstat` command
- `top` command

Allows user to query status of processes. Unix will display information such as:
- process ID, parent process ID
- priority, memory mgmt info, runtime

Unix process monitoring

The Unix `time` command (process execution stats)

`time <process binary file>`

Unix will display the following timing information:

- `user` system elapsed %

Where user is CPU time in user code
- system is CPU time in kernel code
- elapsed is total time in system
- % is percentage of elapsed time that is CPU time