Unix Process Management

Unix Processes

- 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)

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 %d\n", pid);
    else
        printf( "fork returned error code, no child\n" );
}
```
Unix fork system call

- BEFORE fork
  - printf("one\n");
  - pid = fork();
  - printf("two\n");

- AFTER fork
  - printf("one\n");
  - pid = fork();
  - printf("two\n");

  Process A

  Parent process A

  Child process B

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 except if 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**
  - `Printf("hello\n");`
  - `Pid = execl("bin/ls");`
  - `Printf("never\n");`

Process A

- **AFTER exec**

  ![Code for ls]

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( "\nI am the child, running %s.\n", program_name );

        /* never returns if all goes well */
        error_code = execlp( program_name,
            program_name,
            "-l", (char*)0 );
        printf( "Error running ls, execlp returned %d.\n", error_code );
        exit(0);
    }

    // This code never executed.
}
```

Interrupts
Unix process creation (cont)

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