Process Control

15-123
Systems Skills in C and Unix
Process Management

• **A process**
  - is an instance of a program that is currently running.
  - Example: an executing C program

• **A uni processor or a single core system**
  - A system with a single processor
  - A single **processor** can typically executes multiple **processes**

• **A call to a program spawns a process.**
  - If a mail program is called by n users then n processes or instances are created and executed by the unix system.

• Many operating systems including windows and unix **executes many processes** at the same time.
  - Shared systems
Process Status

- When a program is called, a **process is created** and a **process ID** is issued. The process ID is given by the function `getpid()` defined in `<unistd.h>`.

  The prototype for `pid()` is given by

  ```
  #include <unistd.h>
  pid_t getpid(void);
  ```

- `ps` command lists all the current processes

  ```
  > ps
  ```

<table>
<thead>
<tr>
<th>PID</th>
<th>TTY</th>
<th>TIME</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>10150</td>
<td>pts/16</td>
<td>00:00:00</td>
<td>csh</td>
</tr>
<tr>
<td>31462</td>
<td>pts/16</td>
<td>00:00:00</td>
<td>ps</td>
</tr>
</tbody>
</table>
ps command options

> ps -a
> ps -l
> ps -al

Information provided by each process may include the following.

**PID**  The process ID in integer form
**PPID** The parent process ID in integer form
**STAT** The state of the process
**TIME** CPU time used by the process (in seconds)
**TT**   Control terminal of the process
**COMMAND** The user command that started the process
More on processes

Sample Code

- printf("The current process %d \n", getpid());
- printf("The parent process is %d \n", getppid());
- printf("The owner of this process has uid %d \n", getuid());
- sleep(1);

- **Background Processes**
  - run a C program in the background
    - > ./a.out &
  - Ideal for long jobs
Concurrent processes are interleaved
- A way to organize jobs to increase performance
- Concurrency can be enabled
  - when accessing slow I/O devices
- Concurrency Can also be controlled from programmer level
  - Mix I/O and other operations
In Multi-core machines, concurrency is
- True parallelism @ OS level
Application level concurrency

- Exploited by “concurrent programs”
- Three basic approaches to building concurrent applications
  - Multiple Processes
    - Separate virtual address spaces
    - Communicate via IPC
  - I/O multiplexing
    - Application scheduling logical flows in a context of a single process
  - Threads
    - Logical flows that runs in the context of a single process called parent
    - Separate stack space for each thread
How to build concurrency in your program

- Using system calls
  - fork(), exec(), waitpid(), exit()
- Concurrency examples
  - Serving clients in a network
    - Accept requests by client
    - Create threads to handle each client
  - A broadcasting application
    - Data distributed to all nodes in a network by using multiple threads
Creating a child thread

- **fork( )**
  - `#include <unistd.h>`
  - `pid_t fork(void);`
  - `fork` creates a new child process exactly identical to the parent.
  - That is, Child gets an exact copy of the parent.
    - inherits state
  - Child gets a unique process ID.
  - Child also inherits parents file descriptors and refer to the same open files.
Forking new Processes

- Calling `fork()`
  - creates a child process which is exactly identical to the parent process
  - The value zero gets returned to the child and PID gets returned to the parent.
- An example
  ```c
  if (fork() == 0) {
    printf("This is a message from the child\n");
  }
  else { printf("This is a message from the parent\n");}
  ```
- If the fork process is failed, no child process is created and fork returns -1.
  ```c
  int PID = fork();
  if (PID == -1) printf("the process creation failed\n");
  ```
```c
int A[]={1,2,3,4,5,6};
int sum=0, pdt=1, PID, i;
if ((PID=fork())==0){
    for (i=0;i<6;i++) sum += A[i];
    printf("This is child process computed sum %d \n", sum);
}
if (PID <0) {
    fprintf(stderr,"problem creating a process \n");
}
if (PID >0) {
    for (i=0;i<6;i++) pdt *= A[i];
    printf("The parent process completed the product %d \n", pdt);
}

• What is the output?
```
Being Bad
fork bomb
Server-Client Architectures

Client 1

Client 2

Server
More about processes

- Parent and child processes share state information
  - Gets a copy of the state variables
- Parent and children have their own address spaces
  - One process cannot overwrite another
- Drawbacks
  - Hard to share state information
    - However waitpid and signals can send small messages to processes running on the same host
  - Have to use explicit IPC
    - to share information on different hosts
Other Process Management Commands

- **exec()** [many variations of this]
  - See next slide
- **wait()**
  - #include `<sys/wait.h>`
    ```c
    pid_t wait(int *stat_loc);
    ```
    - Suspends the execution of the calling thread until a child has returned
  - **pid_t waitpid(pid_t pid, int *stat_loc, int options);**
    - If pid>0, this requests the status of a child process
    - Options defined in `<sys/wait.h>`
- **exit()**
  - #include `<stdlib.h>`
    ```c
    void exit(int status);
    ```
    - Status can be EXIT_SUCCESS, EXIT_FAILURE or any other value
    - 8 Least significant bits available to a calling process
    - Value can be retrieved by wait
Executing another process

- **execl** --- takes the path name of a binary executable as its first argument, the rest of the arguments are the command line arguments ending with a NULL.
  - **Example**: `execl("./a.out", NULL)`

- **execv** – takes the path name of a binary executable as its first argument, and an array of arguments as its second argument.
  - **Example**: `static char* args[] = {"", "cat.txt", "test1.txt", NULL};
    execv("/bin/cp", args);`

- **execlp** --- same as execl except that we don’t have to give the full path name of the command.
  - `execlp("ls", NULL)`
Writing a (fake) Shell

```c
int PID; char cmd[256];
while (1) {
    printf("cmd: "); scanf("%s", cmd);
    if ( strcmp(cmd,"e") == 0)
        exit(0);
    if ((PID=fork()) > 0)
        wait(NULL);
    else if (PID == 0) /* child process */
        { execlp (cmd,cmd,NULL);
          fprintf (stderr, "Cannot execute %s\n", cmd);
          exit(1);
        }
    else if ( PID == -1)
        { fprintf (stderr, "Cannot create a new process\n");
          exit (2);
        }
}
```
Wait Examples

wait, waitpid - wait for a child process to stop or terminate

```c
#include <sys/wait.h>

pid_t wait(int *status);

pid_t waitpid(pid_t pid, int *status, int options);
```

It returns the PID of the child and the exit status gets placed in status.

```c
main() {
    int child_status, pid, pidwait;
    if ((pid = fork()) == 0) {
        printf("This is the child!\n");
    }
    else {
        pidwait = wait(&child_status);
        printf("child %d has terminated\n", pidwait);
    }
    exit();
}
```
Coding Examples