Computer Science CSCI 251

Systems and Networks

Dr. Peter Walsh
Department of Computer Science
Vancouver Island University
peter.walsh@viu.ca

Process Model

- Program
  - a executable program is a list of machine language instructions and data
- Process
  - a process is the memory space and settings in which the program runs
- Shell
  - a shell is a program that manages processes and runs programs
  - e.g., sh, bash, csh and zsh

Process Model cont.

- Shell Environment Variables
  - shell environment variables contain information about a process
- Process PID
  - each process is assigned an integer process identifier (PID)
- `init` Process
  - the `init` process is the first process created on-boot and is assigned PID 1
  - all other processes are considered descendents of the `init` process

Shell Alternatives

```
peter@cobra:~$ echo $SHELL
/bin/bash
peter@cobra:~$ ls -l /bin/bash
-rwxr-xr-x 1 root root 1183448 Jun 18  2020 /bin/bash
peter@cobra:~$ which sh
/usr/bin/sh
peter@cobra:~$ ls -l /usr/bin/sh
lrwxrwxrwx 1 root root 4 Jun  8  2020 /usr/bin/sh -> dash
```

Shell Alternatives cont.

```
peter@cobra:~$ which csh
/usr/bin/csh
peter@cobra:~$ ls -l /usr/bin/csh
lrwxrwxrwx 1 root root 21 Sep 23 18:18
/usr/bin/csh -> /etc/alternatives/csh
peter@cobra:~$ csh
cobra:~% echo $shell
/bin/tcsh
cobra:~% ls -l /bin/tcsh
-rwxr-xr-x 1 root root 447896 Jul 16  2019 /bin/tcsh
```

Shell Commands

- bash
  - `echo $SHELL` # show shell path
  - `echo $PPID` # show parent process PID
- csh
  - `echo $shell` # show shell path
- Common to bash and csh
  - `ps aux` # show processes for all users
  - `ps -C "command"` # search for a process by # its "command" name
  - `echo $$` # show current process PID
  - `top` # show process statistics
  - `kill -9 PID` # kill a process

Shell Scripts

```
#!/usr/bin/sh

# dirBack

#.dirBak.sh dir (relative to /home/peter)

[ ! -d "/home/peter/$1" ] &&
echo "Directory /home/peter/$1 not found" && exit
rm -f p.tar
tar cvf p.tar /home/peter/$1
fn2=$(date '+%Y-%m-%d-%H-%M-%S')
fn1=$(echo $1 | sed -e 's/\//-/g')
fn=$fn1"+"$fn2
#fn=$(date +%F)
ex=".tar"
fd="/home/peter/Backups/"
target=$fd$fn$ex
echo "Moving tar file to pletus:~peter/Backups/"
scp -r p.tar  peter@pletis:"$target"
rm -f p.tar
```

Shell Scripts cont.

```
#!/usr/bin/csh

# script to forward attachments to otter students

#./batchEmail.csh

set PREFIX = /home/peter/Courses/261/Submissions/Task6/graded
set SOL=task6.pdf.pdf

foreach F ($PREFIX/*)
   if ((${F:t} != pwalsh) && (${F:t} != Report)) then
      if ( -e "$PREFIX/${F:t}/$SOL" ) then
         echo Exists
         mpack -s "CSCI 261 task6" -a $PREFIX/${F:t}/$SOL
         ${F:t}@otter.csci.viu.ca
      else
         echo PROBLEM with ${F:t}
      endif
   endif
end
```

Process Creation (API)

- Fork
  - fork creates a child process that is a clone of its parent
- Exec
  - "boots" the child with a different executable image
- Wait
  - waits for a child to complete

Fork and Exec Example

```
#!/usr/bin/perl

# Here is an example of a program segment which forks and execs
#       "ls -l > dir.out":


       if (($pid = fork()) == 0) { # I am the child
          exec ("ls -l > dir.out");
          print ("Could not exec: errno is $!\n");
          exit (0);
       } elsif ($pid > 0) { # I am the parent
         print ("Parent PID = ", $$, "\n");
         print ("Child PID = ", $pid, "\n");
         $dead_child = wait;
         print ("Dead Child PID = ", $dead_child, "\n");

       } else {
         print ("Could not fork: errno is $!\n");
       }
```

Fork and Exec Example cont.

```
# Fork returns the child pid to the parent process, 0 to the child
# process, or undef if the fork is unsuccessful.

# Wait waits for a child process to terminate and returns the pid of
# the deceased process, or -1 if there are no child processes.

# In Unix, a process can have children created by fork or similar
# system calls. When the child terminates a SIGCHLD signal is sent
# to the parent.

# When a child process terminates before the parent has called wait,
# the kernel retains some information about the process to enable
# its parent to call wait later. Because the child is still
# consuming system resources but not executing it is known as
# zombie process.
```

Unexpected Events

Exceptions and interrupts are unexpected events that disrupt the normal flow of instruction execution.

- Interrupt
  - generated by an external hardware device (typically)
  - adjudicated by the processor hardware
  - handled by the kernel
- Exception
  - generates an internal signal
  - adjudicated by the kernel
  - handled by processes
  - form of inter process communication (IPC)

Signals

- Command Line Examples
  - CTRL-C, CTRL-Z, kill -9 PID
- Process options on receipt of a signal
  - ignore the signal
  - terminate with/without a core dump
  - call a handler function

```
Name            Default Action      Description           ID
SIGINT          Quit                Interrupt             2
SIGKILL         Dump                Kill                  9
SIGCHLD         Ignore              Child status change   20

kill -l will generate the comple list of SIGNALS
```

Signal Handler Example

```
#!/usr/bin/perl

# Here is an example of a program segment which
# catches the signal INT

      $SIG{INT} = sub {leaveScript();};

      sub leaveScript {
         print("\nShutdown Now !!!!! \n");
         exit();
      }

      while (1) {
      }
```

Daemons and Jobs

- Daemon
  - a process that starts at system startup
- Job
  - program that is started interactively by the shell
  - shell can run one job in the foreground and many jobs in the background
  - jobs can be suspended (SIGSTOP)
  - jobs can be restored (SIGCONT)
  - jobs are identified by their job ID (JID)

Foreground/Background Examples

```
sleep 5     job runs in foreground and terminates in 5 sec

sleep 5&    job runs in background and terminates in 5 sec

sleep 5
CTRL-Z      job is suspended to the background

fg JID      brings job JID back to the foreground

bg JID      jobs previously suspended in the background
            can be started in the background
            (job receives a SIGCONT signal)
```