Edgar Villasenor
WSU ID: 11536698
CptS 360
22 September 2020

Lab 3 Pre Work

1. READ List: Chapter 3: 3.1-3.5

What's a process? (Page 102)

A process is an execution of an image. Processes have access to the image's state and have access to the program's code and activity.

Each process is represented by a PROC structure.
Read the PROC structure in 3.4.1 on Page 111 and answer the following questions:

What's the meaning of:
  pid, ppid? Process ID and parent process ID
  status ? Process status, i.e. free, ready, etc.
  priority ? Scheduling priority, a scheduler determines which process is currently running
  event ? event value to sleep on
  exitCode ? exit value

READ 3.5.2 on Process Family Tree. What are the
  PROC pointers child, sibling, parent used for? The process family tree is implemented as a binary tree by a pair of child and sibling PROC pointers.

2. Download samples/LAB3pre/mtx. Run it under Linux.
  MTX is a multitasking system which simulates process operations of
      fork, exit, wait, sleep, wakeup
  in a Unix/Linux kernel

```
/********** A Multitasking System ************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "type.h"    // PROC struct and system constants

// global variables:
PROC proc[NPROC], *running, *freeList, *readyQueue, *sleepList;

running    = pointer to the current running proc
freeList   = a list of all FREE PROCs
readyQueue = a priority queue of procs READY to run
sleepList  = a list of SLEEP procs, if any.

Run mtx. It first initialize the system, creates an initial process P0.
P0 has the lowest priotiry 0, all other processes have priority 1

Ater initialization,
    P0 forks a child prcoess P1, switch process to run P1.
```

The display looks like the following

```
------------------------------------------------------------------------------
Welcome to KCW's Multitasking System
1. init system
freeList = [0 0]->[1 0]->[2 0]->[3 0]->[4 0]->[5 0]->[6 0]->[7 0]->[8 0]->NULL
2. create initial process P0
freeList = [1 0]->[2 0]->[3 0]->[4 0]->[5 0]->[6 0]->[7 0]->[8 0]->NULL
init complete: P0 running
3. P0 fork P1
4. P0 switch process to P1
P0: switch task
proc 0 in scheduler()
readyQueue = [1 1]->[0 0]->NULL
next running = 1
proc 1 resume to body()

proc 1 running: Parent=0 childList = NULL
freeList  = [2 0]->[3 0]->[4 0]->[5 0]->[6 0]->[7 0]->[8 0]->NULL
readQueue = [0 0]->NULL
sleepList = NULL
input a command: [ps|fork|switch|exit|sleep|wakeup|wait] :
------------------------------------------------------------------------------
```

3. COMMANDS:

ps : display procs with pid, ppid, status; same as ps in Unix/Linux

fork : READ kfork() on Page 109: What does it do?

Creates a child process and returns the child pid. The new proc goes in to the ready queue.

switch : READ tswitch() on Page 108: What does it do?

Implements process context switching. When one process goes in, another emerges.

exit : READ kexit() on Page 112: What does it do?

Causes termination of a process and switches context

sleep : READ ksleep() on Page 111: What does it do?

It causes the proc (which needs a resource that is unavailable) to sleep.

wakeup : READ kwakeup() on Page 112: What does it do?

Wakes up the process that were sleeping on a particular event

wait : READ kwait() on Page 114: What does it do?

Waits for a zombie child process.

---------- REQUIREMENTS ----------

4. Step 1: test fork

While P1 running, enter fork: What happens?

P2 is ready to run and is placed into the ready queue

Enter fork many times;

How many times can P1 fork?

P1 can fork 7 times

WHY?

There are 9 procs total, but only 7 procs that are able to be used as children for P1.

Enter Control-c to end the program run.

5. Step 2: Test sleep/wakeup

Run mtx again.
While P1 running, fork a child P2;
Switch to run P2. Where did P1 go?

P1 goes to the ready queue

WHY?

Only 1 proc may be running at a single time.

P2: Enter sleep, with a value, e.g.123 to let P2 SLEEP.
What happens?

P2 is in the sleep list and P1 is the running proc

WHY?

Only one proc can run at a time

Now, P1 should be running. Enter wakeup with a value, e.g. 234
Did any proc wake up?

No

WHY?

P2 should only wakeup for value 123

P1: Enter wakeup with 123
What happens?

There's nothing in the sleep list

WHY?

P2 wakes up on value 123

6. Step 3: test child exit/parent wait

When a proc dies (exit) with a value, it becomes a ZOMBIE, wakeup its parent.
Parent may issue wait to wait for a ZOMBIE child, and frees the ZOMBIE

Run mtx;
P1: enter wait; What happens?

System throws error

WHY?

P1 has no child process

CASE 1: child exit first, parent wait later

P1: fork a child P2, switch to P2.
P2: enter exit, with a value, e.g. 123 ==> P2 will die with exitCode=123.
Which process runs now?

P1 is running

WHY?

P2 is now a zombie process

enter ps to see the proc status: P2 status = ?

P2 is now a zombie process

(P1 still running) enter wait; What happens?
P1 is still running but it's child list is empty
enter ps; What happened to P2?
P2 is a free process

CASE 2: parent wait first, child exit later
P1: enter fork to fork a child P3
P1: enter wait; What happens to P1?
P1 is sleeping
WHY?
Because P3 is running
P3: Enter exit with a value; What happens?
P1 is running and P3 is free
P1: enter ps; What's the status of P3? Free WHY? Calling exit freed the process

7. Step 4: test Orphans

When a process with children dies first, all its children become orphans.
In Unix/Linux, every process (except P0) MUST have a unique parent.
So, all orphans become P1's children. Hence P1 never dies.

Run mtx again.
P1: fork child P2, Switch to P2.
P2: fork several children of its own, e.g. P3, P4, P5 (all in its childList).
P2: exit with a value.
P1 should be running WHY?
Because P2's children are now orphaned to P1
P1: enter ps to see proc status: which proc is ZOMBIE? P2 is zombie proc
What happened to P2's children? They are now P1's children
P1: enter wait; What happens? P2 is free
P1: enter wait again; What happens? P1 sleeps and P3 is running WHY? P1 will wait for a zombie before it starts running again

How to let P1 READY to run again? Exit the current process