Joseph Steeb
9/23/2021

LAB3pre Work: Processes in an OS Kernel
DUE: 9-23-2021
Answer questions below. Submit a (text-edit) file to TA

1. READ List: Chapter 3: 3.1-3.5

What's a process? (Page 102) A process is a task that the operating system must complete, each process is an image that is to be executed.

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, Parent process ID number
status? Current status of the process including FREE/READY
priority? Process scheduling priority
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? First child PROC pointer, sibling PROC pointer, parent PROC pointer.

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

```
/*********** 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 that are 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
init complete: P0 running

3. P0 fork P1 : enter P1 into readyQueue

4. P0 switch process to run 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] :

---

5. 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 task and enters it into the readyQueue.
switch : READ tswitch() on Page 108: What does it do? acts as a process switch box, where one process goes in and, in general, another process emerges.
exit : READ kexit() on Page 112: What does it do? Called when a process is terminated, and handels deallocating memory and terminating child processes.
Sleep : READ ksleep() on Page 111: What does it do? Takes process out of readyQueue, and allows it to “go to sleep”.

wakeup : READ kwakeup() on Page 112: What does it do? Adds a sleeping process to the readyQueue, “waking it up”.
wait : READ kwait() on Page 114: What does it do? Waits for zombie processes and adds them back into freeList.

---------- TEST REQUIREMENTS ----------
6. Step 1: test fork
While P1 running, enter fork: What happens? Proc 2 is taken of the freeList and added to the ready queue

Enter fork many times;
How many times can P1 fork? 7 times WHY? Because that is the number of processes in the freeList

Enter Control-c to end the program run.

7. Step 2: Test sleep/wakeup
Run mtx again.
While P1 running, fork a child P2;
Switch to run P2. Where did P1 go? P1 is put into the ready queue WHY? Because the processor can only execute one process at a time, proc 1 must be moved to ready queue in order for proc 2 to execute.

P2 running : Enter sleep, with a value, e.g.123 to let P2 SLEEP.
What happens? Proc 1 now runs, and proc 2 is added to the sleep list WHY? Because in the previous step proc 1 was put on the ready queue, and now that proc 2 is no longer running, the first element of the ready queue(proc 1) will take its place.

Now, P1 should be running. Enter wakeup with a value, e.g. 234
Did any proc wake up? No WHY? Because the wake up value 234 was not the same as the sleep value 123.

P1: Enter wakeup with 123
What happens? Proc 2 added back into the ready queue WHY? Because wakeup was called with the proper value.

8. 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? A wait error occurs WHY? Because the proc contains no children.

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? Proc1 WHY? Because it was next on the ready queue.
enter ps to see the proc status: P2 status = ? P2 is now a zombie.

(P1 still running) enter wait; What happens? Proc 2 is now at the back of the ready queue.
enter ps; What happened to P2? It is now free and it's ppid is 1.

CASE 2: parent wait first, child exit later

P1: enter fork to fork a child P3
P1: enter wait; What happens to P1? Goes to sleep WHY? Because it has no children.
P3: Enter exit with a value; What happens? proc 1 is now running.
P1: enter ps; What's the status of P3? FREE WHY? _____

9. 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 it is next in the ready que.
P1: enter ps to see proc status: which proc is ZOMBIE? Proc2
What happened to P2's children? They stay in the childList behind zombie proc2.
P1: enter wait; What happens? pro2 added to the back of free list, proc 3 is now child of proc1.
P1: enter wait again; What happens? Proc 1 goes to sleep WHY? Because no zombie nodes exist.

How to let P1 READY to run again? Exit the current proc(proc3).