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)

**Process is the execution of an image, in operating system also tasks.**

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: process ID**

**ppid: parent process pid**

**status: PROC status=FREE|READY, etc.**

**priority: 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?

**The process family tree is implemented as a binary tree by a pair of child and sibling pointers in each PROC.**

**Child points to the first child of a process and sibling points to a list of other children of the same parent. For convenience, each PROC also uses a parent pointer to point at its parent.**

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?

**The kfork() function creates a child task and enters it into the readyQueue.**

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

**The tswitch() function implements process context switching.**

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

**Issues \_exit(value) system call to execute kexit(value) in the OS kernel**

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

**Lets a process go to sleep.**

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

**Call kwakeup(event), which wakes up ALL the processes sleeping on the event value. If no process is sleeping on the event, kwakeup() has no effect**

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

**Call the kernel function to wait for a ZOMBIE child process. Also releases the ZOMBIE child PROC back to the freeList for reuse.**

---------------------------------------------------------------------------

------------------------ TEST REQUIREMENTS ---------------------------------

6. Step 1: test fork

While P1 running, enter fork: What happens?

**Creates a child task for P1 and enters it into the readyQueue.**

Enter fork many times;

How many times can P1 fork? WHY?

**Only one time. Because in binary tree, fork will create the child only once**.

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? WHY?

**Waiting until P2 is done. The reason is computer multitasking.**

P2 running : Enter sleep, with a value, e.g.123 to let P2 SLEEP.

What happens? WHY?

**The P2 will stop until 123(time). The sleep command Lets a process go to sleep**.

Now, P1 should be running. Enter wakeup with a value, e.g. 234

Did any proc wake up? WHY?

**No, because the wakeup value not equal to the sleep value.**

P1: Enter wakeup with 123

What happens? WHY?

**P2 continued running, command wakes up ALL the processes sleeping on the event 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? WHY?

**P1 has to be in wait state if no child terminates. Because it releases the ZOMBIE child PROC back to the freeList for reuse.**

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? WHY?

**Run P3, because already fork the child, then if P2 exit, the binary tree would go down.**

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

(P1 still running) enter wait; What happens?

**Create the zombie and blocked.**

enter ps; What happened to P2?

**View information related to the P2 processes**

CASE 2: parent wait first, child exit later

P1: enter fork to fork a child P3

P1: enter wait; What happens to P1? WHY?

**Wait for any child process to end. Because it suspends the execution of the calling process until any one child process exits.**

P3: Enter exit with a value; What happens?

**When value is equal, exit.**

P1: enter ps; What's the status of P3? WHY?

**Exit. Because the child will exit after parent wait.**

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?

**P1 is parent progress.**

P1: enter ps to see proc status: which proc is ZOMBIE?

**P2**

What happened to P2's children?

**Orphaned child process is adopted**

P1: enter wait; What happens?

**The orphaned child process is adopted.**

P1: enter wait again; What happens? WHY?

**Nothing happen, because parent process is already killed.**

How to let P1 READY to run again?

**Using the screen command?**