Refo Yudhanto

pre Lab 3

/\*\*\*\*\*\*\*\*\*\*\* t.c file of A Multitasking System \*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include "string.h"

#include "wait.c" // include wait.c file

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

kfork() creates a child process; returns child pid.

When scheduled to run, child PROC resumes to body();

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int body(), tswitch(), do\_sleep(), do\_wakeup(), do\_exit(), do\_switch();

int do\_kfork(), do\_wait();

char \*myName = "Refo Yudhanto";

// initialize the MT system; create P0 as initial running process

int init()

{

int i;

PROC \*p;

for (i=0; i<NPROC; i++){ // initialize PROCs

p = &proc[i];

p->pid = i; // PID = 0 to NPROC-1

p->status = FREE;

p->priority = 0;

p->next = p+1;

}

proc[NPROC-1].next = 0;

freeList = &proc[0]; // all PROCs in freeList

readyQueue = 0; // readyQueue = empty

sleepList = 0; // sleepList = empty

// create P0 as the initial running process

p = running = dequeue(&freeList); // use proc[0]

p->status = READY;

p->priority = 0;

p->ppid = 0; // P0 is its own parent

printList("freeList", freeList);

printf("init complete: P0 running\n");

}

int menu()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" ps fork switch exit jesus sleep wakeup wait \n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

int do\_ps()

{

int i;

PROC \*p;

printf("PID PPID status\n");

printf("--- ---- ------\n");

for (i=0; i<NPROC; i++){

p = &proc[i];

printf(" %d %d ", p->pid, p->ppid);

if (p == running)

printf("RUNNING\n");

else

printf("%s\n", status[p->status]);

}

}

int do\_jesus()

{

int i;

PROC \*p;

printf("Jesus perfroms miracles here\n");

for (i=1; i<NPROC; i++){

p = &proc[i];

if (p->status == ZOMBIE){

p->status = READY;

enqueue(&readyQueue, p);

printf("raised a ZOMBIE %d to live again\n", p->pid);

}

}

printList("readyQueue", readyQueue);

}

int body(char\* myname) // process body function

{

int c;

char cmd[64];

if(running->pid == 1){ //set global P1 ptr to P1

P1 = running;

}

printf("proc %d starts from body() ", running->pid);

printf("myname = %s\n", myName);

while(1){

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("proc %d running: parent=%d\n", running->pid,running->ppid);

printChild("ChildList", running->child);

printList("readyQueue", readyQueue);

printSleep("sleepList ", sleepList);

menu();

printf("enter a command : ");

fgets(cmd, 64, stdin);

cmd[strlen(cmd)-1] = 0;

if (strcmp(cmd, "ps")==0)

do\_ps();

if (strcmp(cmd, "fork")==0)

do\_kfork();

if (strcmp(cmd, "switch")==0)

do\_switch();

if (strcmp(cmd, "exit")==0)

do\_exit();

if (strcmp(cmd, "jesus")==0)

do\_jesus();

if (strcmp(cmd, "sleep")==0)

do\_sleep();

if (strcmp(cmd, "wakeup")==0)

do\_wakeup();

if (strcmp(cmd, "wait")==0)

do\_wait();

}

}

int kfork()

{

int i;

PROC \*p = dequeue(&freeList);

if (!p){

printf("no more proc\n");

return(-1);

}

/\* initialize the new proc and its stack \*/

p->status = READY;

p->priority = 1; // ALL PROCs priority=1, except P0

p->ppid = running->pid;

/\*\*\*\*\*\*\*\*\*\*\*\* new task initial stack contents \*\*\*\*\*\*\*\*\*\*\*\*

kstack contains: |retPC|eax|ebx|ecx|edx|ebp|esi|edi|eflag|

-1 -2 -3 -4 -5 -6 -7 -8 -9

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (i=1; i<10; i++) // zero out kstack cells

p->kstack[SSIZE - i] = 0;

p->kstack[SSIZE-1] = (int)body; // retPC -> body()

p->ksp = &(p->kstack[SSIZE - 9]); // PROC.ksp -> saved eflag

enqueue(&readyQueue, p); // enter p into readyQueue

if(running->child){

PROC \*childPtr = running->child;

while(childPtr->sibling){

childPtr = childPtr->sibling;

}

childPtr->sibling = p;

}

else

running->child = p;

p->ppid = running->pid;

p->parent = running;

return p->pid;

}

int do\_kfork()

{

int child = kfork();

if (child < 0)

printf("kfork failed\n");

else{

printf("proc %d kforked a child = %d\n", running->pid, child);

printList("readyQueue", readyQueue);

}

return child;

}

int do\_switch()

{

tswitch();

}

int do\_exit()

{

kexit(running->pid); // exit with own PID value

}

int do\_sleep()

{

int event;

printf("enter an event value to sleep on : ");

scanf("%d", &event); getchar();

sleep(event);

}

int do\_wakeup()

{

int event;

printf("enter an event value to wakeup with : ");

scanf("%d", &event); getchar();

wakeup(event);

}

int do\_wait(){

int status;

int pid = wait(&status);

printf("proc%d buried zombie child = %d, status = %d\n",running->pid, pid,status);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* main() function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

printf("Welcome to the MT Multitasking System\n");

init(); // initialize system; create and run P0

kfork(); // kfork P1 into readyQueue

while(1){

printf("P0: switch process\n");

while (readyQueue == 0);

tswitch();

}

}

/\*\*\*\*\*\*\*\*\*\*\* scheduler \*\*\*\*\*\*\*\*\*\*\*\*\*/

int scheduler()

{

printf("proc %d in scheduler()\n", running->pid);

if (running->status == READY)

enqueue(&readyQueue, running);

printList("readyQueue", readyQueue);

running = dequeue(&readyQueue);

printf("next running = %d\n", running->pid);

}

/\*\*\* wait.c \*\*\*/

#include "queue.c"

int tswitch();

int sleep(int event)

{

printf("proc %d going to sleep on event=%d\n", running->pid, event);

running->event = event;

running->status = SLEEP;

enqueue(&sleepList, running);

printList("sleepList", sleepList);

tswitch();

}

int wakeup(int event)

{

PROC \*temp, \*p;

temp = 0;

printList("sleepList", sleepList);

while (p = dequeue(&sleepList)){

if (p->event == event){

printf("wakeup %d\n", p->pid);

p->status = READY;

enqueue(&readyQueue, p);

}

else{

   enqueue(&temp, p);

}

}

sleepList = temp;

printList("sleepList", sleepList);

}

int kexit(int exitValue)

{

if(exitValue != 1){

printf("proc %d in kexit(%d)\n", running->pid, exitValue);

running->exitCode = exitValue; //record exitValue in proc's exitCode

running->status = ZOMBIE; //become a ZOMBIE

if(running->child){

PROC \*runChild = running->child; //point to running child

PROC \*p1child = P1->child; //point to p1's child

if(p1child){//give away children, if any, to P1

while(p1child->sibling){

p1child = p1child->sibling;

}

p1child->sibling = runChild; //combine

while(runChild){ //set runChild parent ptr to p1

runChild->ppid = P1->pid;

runChild->parent = P1;

runChild = runChild->sibling;

}

running->child = NULL;

}

else{

}

}

wakeup(running->ppid); //wakeup parent (by parent proc address)

tswitch();

}

else{

}

}

int letgo(PROC \*ptrChild, PROC \*ptrPrevChild){

if(!ptrChild->sibling){

running->child = NULL;

}

else if(running->child == ptrChild){

running->child = ptrChild->sibling;

}

else{

ptrPrevChild->sibling = ptrChild->sibling;

}

}

int freeZomb(PROC \*ptr){

ptr->sibling = NULL;

ptr->parent = NULL;//remove all connection

ptr->ppid = 0;

ptr->status = FREE;

}

int wait(int \*status){

int zomBid;

if(running->child){

PROC \*child = running->child;

PROC \*prev = running;

while(1){

if(child->status == ZOMBIE){ //found zombie child

zomBid = child->pid;

(\*status) = child->exitCode; //copyt ZOMBIE child exitCode to \*status

letgo(child, prev); //unlink

freeZomb(child); //bury

enqueue(&freeList,child); //put into freeList

wakeup(running->ppid);

return zomBid;//return pid

}

if(!child->sibling){

child = running->child;

}

else{

prev = child;

child = child->sibling;

}

sleep(running->pid);

}

}

else{

return -1; //if no child

}

}

#include "type.h"

PROC proc[NPROC]; // NPROC PROCs

PROC \*freeList; // freeList of PROCs

PROC \*readyQueue; // priority queue of READY procs

PROC \*running; // current running proc pointer

PROC \*sleepList; // list of SLEEP procs

PROC \*P1; // to always point at P1

char \*status[ ] = {"FREE", "READY", "SLEEP", "ZOMBIE"};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* queue.c file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int enqueue(PROC \*\*queue, PROC \*p)

{

PROC \*q = \*queue;

if (q == 0 || p->priority > q->priority){

\*queue = p;

p->next = q;

}

else{

while (q->next && p->priority <= q->next->priority)

q = q->next;

p->next = q->next;

q->next = p;

}

}

PROC \*dequeue(PROC \*\*queue)

{

PROC \*p = \*queue;

if (p)

\*queue = (\*queue)->next;

return p;

}

int printList(char \*name, PROC \*p)

{

printf("%s = ", name);

while(p){

printf("[%d %d]->", p->pid, p->priority);

p = p->next;

}

printf("NULL\n");

}

int printSleep(char \*name, PROC \*p)

{

printf("%s = ", name);

while(p){

printf("[%d event=%d]->", p->pid, p->event);

p = p->next;

}

printf("NULL\n");

}

int printChild(char \*words, PROC \*child)

{

printf("%s = ", words);

while(child){

printf("[%d %s]->", child->pid, status[child->status]);

child = child->sibling;

}

putchar('\n');

}