type.h

//

// type.h

// 360Prelab2

//

// Created by Weiren Lai on 1/29/20.

// Copyright © 2020 Weiren Lai. All rights reserved.

//

/\*\*\*\*\*\*\*\*\* type.h \*\*\*\*\*\*\*\*\*\*\*/

#define NPROC 9

#define SSIZE 1024

#define FREE 0 // proc status

#define READY 1

#define SLEEP 2

#define BLOCK 3

#define ZOMBIE 4

**typedef** **struct** proc{

**struct** proc \*next; // next proc pointer

**int** \*saved\_sp; // at offset 4: do NOT alter

**int** pid; // pid = 0 to NPROC-1

**int** ppid; // parent pid

**int** status; // PROC status: FREE|READY|etc

**int** priority; // scheduling priority

**int** event; // event to sleep on

**int** exitCode; // exit code value

**struct** proc \*child; // first child PROC pointer

**struct** proc \*sibling; // sibling PROC pointer

**struct** proc \*parent; // parent PROC pointer

**int** kstack[SSIZE]; // processs stack

}PROC;

PROC proc[NPROC], \*running, \*freeList, \*readyQueue;

PROC \*sleepList; // for sleep()/wakeup()

queue.c

//

// queue.c

// 360Prelab2

//

// Created by Weiren Lai on 1/29/20.

// Copyright © 2020 Weiren Lai. All rights reserved.

//

#include "type.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 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;

}

**return** 0;

}

// remove and return first PROC in queue

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");

**return** 0;

}

wait.c

//

// type.c

// 360Prelab2

//

// Created by Weiren Lai on 1/29/20.

// Copyright © 2020 Weiren Lai. All rights reserved.

//

#include "type.h"

**extern** PROC \*running;

**int** kexit(**int** value)

{

running->exitCode = value;

running->status = ZOMBIE;

tswitch();

**return** 0;

}

/\*\*\*\*\*\*\*\*\* Required PRE-work \*\*\*\*\*\*\*\*\*\*\*/

// 1. Modify kfork() to implement process tree as a BINARY tree

// 2. Implement ksleep() per the algorithm in 3.4.1

**int** ksleep(**int** event)

{

running->event=event;

running->status=SLEEP;

enqueue("sleepList", sleepList);

tswitch();

**return** 0;

}

// 2. Implement kwakeup per the algorithm in 3.4.2

**int** kwakeup(**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);

**return** 0;

}

**int** mexit(**int** value)

{

PROC \*cur;

**int** wakeUp = 0;

**if**(running->pid == 1){

**return** -1;

}

**for** (**int** i = 1; i < NPROC; i++)

{

cur = &proc[i];

**if**(cur -> status != FREE && cur->ppid == running ->pid)

{

cur->ppid = 1;

cur->parent = &proc[1];

wakeUp = 1;

}

}

running -> exitCode = value;

running -> status = ZOMBIE;

wakeup ((**int**)(running->parent));

**if**(wakeUp){

wakeup((**int**)(&proc[1]));

}

tswitch();

**return** 0;

}

// 3. Implement kwait() per the algorithm in 3.5.3

**int** kwait(**int** \*status)

{

**int** childexists = 0;

PROC \*cur;

**while**(1)

{

**for** (**int** i = 1; i < NPROC; i++)

{

cur = &proc[i];

}

**if**(cur->ppid == running -> pid){

**if** (cur -> status != FREE)

{

childexists =1;

}

**if**(cur -> status = ZOMBIE)

{

\*status = cur -> exitCode;

cur -> status = FREE;

enqueue(&freeList, cur);

**return** cur -> pid;

}

}

**if**(!childexists){

**return** -1;

}

ksleep((**int**)(running));

}

}

// 4. Add a "wait" command to let proc wait for ZOMBIE child

**int** wait(**int** \*status)

{

**int** childexists = 0;

PROC \*p;

**while**(1)

{

**for**(**int** i = 1; i <NPROC; i++)

{

p = & proc[i];

**if** (p -> ppid == running -> pid)

{

**if**(p -> status != FREE)

{

childexists = 1;

}

**if**(p -> status == ZOMBIE)

{

\*status = p -> exitCode;

p -> status = FREE;

enqueue(&freeList, p);

**return** p -> pid;

}

}

}

**if**(!childexists){

**return** -1;

}

ksleep((**int**)(running));

}

}

t.c

//

// main.c

// 360Prelab2

//

// Created by Weiren Lai on 1/29/20.

// Copyright © 2020 Weiren Lai. All rights reserved.

//

/\*\*\*\*\*\*\*\*\*\*\* A Multitasking System \*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "type.h"

#include "queue.c"

#include "wait.c"

PROC proc[NPROC], \*running, \*freeList, \*readyQueue;

PROC \*sleepList; // for sleep()/wakeup()

**int** body()

{

**char** command[64];

printf("proc %d resume to body()\n", running->pid);

**while**(1){

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

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

printList("freeList ", freeList);

printList("readQueue", readyQueue);

printf("input a command: [ps|fork|switch|exit] : ");

fgets(command, 64, stdin);

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

**char** \*cmds[] = {"ps", "fork", "switch", "exit", "sleep", "wakeup", "wait", 0};

**int** (\*fptr[])() ={do\_ps(), do\_fork(), do\_switch(), do\_exit(), do\_sleep(), do\_wakeup(), do\_wait()};

**int** findCmd(**char** \*cmds){

**int** i = 0;

**while**(cmds[i])

{

**if**(strcmp(cmds, cmds[i]==0))

**return** i;

i++;

}

**return** -1;

}

**int** index = findCmd(cmds);

**if**(index>=0)

{

fptr[index]();

}

}

}

**int** do\_switch()

{

printf("proc %d switch task\n", running->pid);

tswitch();

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

**return** 0;

}

**int** do\_fork()

{

**int** child = kfork(body);

**if** (child < 0)

printf("kfork failed\n");

**else**{

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

printList("readyQueue", readyQueue);

}

**return** child;

}

**char** \*pstatus[]={"FREE ","READY ","SLEEP ","BLOCK ","ZOMBIE", "RUNNING"};

**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("%s\n", pstatus[5]);

**else**

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

}

**return** 0;

}

**int** do\_exit()

{

**int** value;

PROC \*p;

**if** (running->pid==1){

printf("P1 never dies\n");

**return** 0;

}

printf("proc %d in do\_exit(), enter an exit value : ", running->pid);

scanf("%d", &value);

mexit(value);

**return** 0;

}

**int** kfork(**int** (\*func))

{

PROC \*p;

**int** i;

/\*\*\* get a proc from freeList for child 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; // for ALL PROCs except P0

p->ppid = running->pid;

p->parent = running;

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

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

**for** (i=1; i<10; i++)

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

p->kstack[SSIZE-1] = (**int**)func;

p->saved\_sp = &(p->kstack[SSIZE - 9]);

enqueue(&readyQueue, p);

**return** p->pid;

}

**int** init()

{

**int** i;

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

proc[i].pid = i;

proc[i].status = FREE;

proc[i].priority = 0;

proc[i].next = (PROC \*)&proc[(i+1)];

}

proc[NPROC-1].next = 0;

freeList = &proc[0];

readyQueue = 0;

// create P0 as the initial running process

running = dequeue(&freeList);

running->status = READY;

running->priority = 0;

running->parent = running;

printList("freeList", freeList);

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

**return** 0;

}

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

**int** main()

{

printf("\nWelcome to 360 Multitasking System\n");

init();

printf("P0 fork P1\n");

kfork(body);

**while**(1){

**if** (readyQueue){

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

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);

**return** 0;

}