**Project 1**

**CIS 620**

|  |  |
| --- | --- |
|  | Name: Binu Joyce  Login Id: Bijoyce |

Project 2 – Report

# Introduction

This project is to familiarize ourselves with design and implementation of user-level thread package onl Linux/PC.

There are two parts of this project.

* Part I
* Part II

# Part I

Part I requires us to modify the non-preemptive DIY thread package and make it preemptive. For this we will be using SIGALRM interrupt instead. We will NOT be calling the xthread\_yield() function.

The below code describes the way we have tried to set the timer of 0.02 secs using ualarm() and handler() function. We have called the Signal handler in main.c as that is the main thread created, visible to the OS, and alive for the complete program.

**resched.c**

*/\* resched.c - resched \*/*

#include **<stdio.h>**

#include **<proc.h>**

#include **<signal.h>**

**void** handler();

*/\*------------------------------------------------------------------------*

*\* resched -- find a live thread to run*

*\**

*\*------------------------------------------------------------------------*

*\*/*

**void** resched()

{

*// signal(SIGALRM, handler); // signal*

**register struct** xentry \*cptr; */\* pointer to old thread entry \*/*

**register struct** xentry \*xptr; */\* pointer to new thread entry \*/*

**int** i,next;

*/\* keyword register: register hints the compiler that the given variable*

*\* can be put in register. It is the register's choice to put in a refister*

*\* or not. Generally, compilers do optimizations and put the var in register*

*\* register is faster than memory*

*\* don't use "&var", because var is in register other than memory.*

*\*/*

cptr = &xtab[currxid]; */\* get current thread's address of PCB/Xtab \*/*

next = currxid ; */\* point to current thread's xid \*/*

*/\* find the next thread whose state is XREAD and do context switch\*/*

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

**if**( (++next) >= **NPROC**)

next = 0;

**if**(xtab[next].xstate == **XREADY**) {

xtab[next].xstate = **XRUN**;

xptr = &xtab[next];

currxid = next;

*// signal(SIGALRM, handler);*

ualarm(20000, 0); *// interrupt every 20 ms after the new thread is running*

printf(**"in resched(): currxid:{%d} state{%d} \n"**, currxid, xtab[currxid].xstate); *//testing*

ctxsw(cptr->xregs,xptr->xregs);

*//printf("in resched-ctxsw "); //testing*

**return**;

}

}

printf(**"XT: no threads to run!\n"**);

exit(0);

}

**void** handler() {

sigset\_t set; *// signal data structure set*

sigemptyset(&set); */\* initilaize the value of the set*

*\* the value of 32-bit blocked list => 0:*

*\* don' block any types of signal.*

*\*/*

sigaddset(&set, **SIGALRM**); *// add SIGALRM to variable set*

*// set bit position of SIGALRM as 1*

*// Unblock signal SIGALRM as specified in variable set*

**if**(sigprocmask(**SIG\_UNBLOCK**, &set, 0) < 0) {

perror(**"sigprocmask"**);

exit(1);

}

printf(**"in handler(). currxid: %d\n"**, currxid); *//testing*

*// change the current thread's state from XRUN to XREADY*

xtab[currxid].xstate = **XREADY**;

resched();

}

**main .c**

#include **<stdio.h>**

#include **<proc.h>**

#include **<signal.h>**

**extern void** handler();

**extern void** xmain(); */\* declare xmain function which is in another file \*/*

**struct** xentry xtab[10]; */\* define global var xtab as an array, size 10 \*/*

**int** currxid = 0; */\* the first thread's PCB information/Xentry stored in*

*\* xtab[xurrxid] => xtab[0].*

*\*/*

**void** main(**int** argc, **char** \*argv[])

{

**register struct** xentry \*xptr; */\* xptr can be put in register in CPU \*/*

**struct** xentry m;

**int** i;

**int** xidxmain;

*/\* initialize the xtab/PCB, inlucding xid, stack(xbase, xlimit), state \*/*

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

xptr = &xtab[i];

xptr->xid = i;

xptr->xlimit = (WORD) malloc(**STKSIZE**);

xptr->xbase = xptr->xlimit + **STKSIZE** - **sizeof**(WORD);

xptr->xstate = **XFREE**;

}

*/\* the first thread runs user's xmain with id 0\*/*

*// xidxmain = xthread\_create(xmain\_part1, 2, argc, argv); // test xmain\_part1*

xidxmain = xthread\_create(xmain, 2, argc, argv); *// test xmain*

xtab[xidxmain].xstate = **XRUN**;

signal(**SIGALRM**, handler); *// it is better to place signal in main funciton.*

ctxsw(m.xregs, xtab[xidxmain].xregs);

*/\* never be here \*/*

}

# Part II

Part 2 of the project is regarding xthread library enhancement by adding the event mechanism. This feature has been implemented by using three functions (xthread\_init\_ev() xthread\_wait\_ev(), xthread\_set\_ev() ) and a struct datatype xthread\_event\_t.

The way it works is once the “event” (ep – event pointer) is initialised then if the event has “NOT OCCURED” then the active running thread’s state is changed from XRUN to XEVENT mimicking the WAIT state. After this, once the xthread\_set\_ev operation is performed by a thread then all the treads that were loaded to the “waiting queue” are allowed to continue processing for their slot times os 0.02 secs each and the xthread\_event\_t’s event indicator (a bool variable) is set to “NOT OCCURRED”

So, using this event mechanism we able to move a thread from “XRUN” state to “XEVENT” (wait state) and then to XREADY state.

**proc.h**

**struct** xthread\_event\_t {

**bool** occurred;

**int** Q[**NPROC**];

};

**extern struct** xthread\_event\_t e;

**extern int** qi; *// Q[qi]*

**extern bool** isEmpty;

**extern int** cid;

xthread\_init\_ev()

xthread\_wait\_ev()

xthread\_set\_ev()

**event.c**

#include **<proc.h>**

#define **NOT\_OCCURRED false**

#define **OCCURRED true**

*// your code will not call xtread\_yield() and ctxsw()*

*// just use resched()*

**void** xthread\_init\_ev(**struct** xthread\_event\_t \* ep) {

**int** remain = ualarm(0, 0);

printf(**"in init\_ev. remain: %d"**, remain);

ep->occurred = **NOT\_OCCURRED**;

ualarm(remain);

}

**void** xthread\_wait\_ev(**struct** xthread\_event\_t \* ep) {

**int** remain = ualarm(0, 0); *// disable time interrupt*

printf(**"in init\_ev. remain: %d"**, remain);

**if**(ep->occurred == **NOT\_OCCURRED**) { *// if event is not occurred,*

ep->Q[qi++] = cid; *// insert/put the current thread into the wait Q*

isEmpty = **false**;

xtab[cid].xstate = **XEVENT**; *// change the current thr's state = XEVENT*

printf(**"\n** **\*\*\*\*\*\*in wait\_ev of {%d} is {%d}\n"**, ep->Q[qi], xtab[ep->Q[qi]].xstate ); *//for testing*

resched(); *// this thr will be back to ready state when someone call it*

**return**;

} **else** {

ep->occurred = **NOT\_OCCURRED**;

}

ualarm(remain); *// enable timer at ending;*

}

**void** xthread\_set\_ev(**struct** xthread\_event\_t \* ep) {

**int** remain = ualarm(0, 0);

**if**(isEmpty == **false**) { *// ep->Q is not empty*

isEmpty = **true**;

printf(**"in set\_ev of {%d} is {%d}\n"**, ep->Q[qi], xtab[ep->Q[qi]].xstate ); *//for testing*

ep->occurred = **NOT\_OCCURRED**;

**while**(qi >= 0) {

xtab[ep->Q[qi]].xstate = **XREADY**;

ep->Q[qi] = -1;

}

} **else** { *// if the Q is empty, the event will be set as OCCURRED*

ep->occurred = **OCCURRED**;

}

ualarm(remain);

}

# Experience in debugging

Implementation of this project both part 1 and part 2 were tough. We struggled with the placement of Signal Handler function (signal(**SIGALRM**, handler) and were intially getting segmentation fault errors

The second issue was with the implementation of Part2. The way we had set the functionality was causing the thread that thread that calls the xthread\_wait\_ev got stuck in a continuous loop inside moving from reshed() to hadler() and back to resched(). The issue was with the while loop in the xthread\_set\_ev.

# Inconsistent State (when not using SIGALRM)

???

# Thread State Transition Diagram:

Thread terminates

New thread create (XFREE)

Schedular picks. Resched()

Timer expires

Ualarm(), SIGALRM, resched()

Xthread\_wait\_ev()

from event.c . (Threads wait in a queue)

Xthread\_set\_ev()

from event.c