CIS 620 Project 2 Fall 2018

(Due: Oct. 22)

The purpose of this project is to familiarize yourself with the design and implementation issues of a user-level thread package on PC/Linux. You need to use the `tar` command with the options `xvfz` to uncompress and extract files from ~cis620s/pub/xt.tar.gz to your working directory.

Part I

The thread package which we discussed in the class is non-preemptive. That is, a thread can run to completion unless it yields the control to other threads. For the first part of this assignment, you need to make the threads preemptive. You can use `signal` and `ualarm` to provide a clock interrupt every 0.01 second. When the clock interrupt occurs, the interrupt handler suspends the current running thread and finds a runnable thread to run. On PC/Linux, you need `sigemptyset`, `sigaddset`, and `sigprocmask` to unblock the SIGALRM signal to allow the next SIGALRM delivered. Add more comments to the source code.

Part II

You also need to enhance the thread library with the message passing mechanism. The mailboxes are declared as variables with a data type `xthread_mbox_t`. Each message box can hold only one positive integer value. It also has a queue of threads waiting for a message.

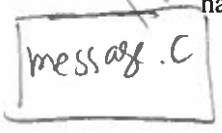
You have to implement the following four thread functions:

- `int xthread_init_mbox(xthread_mbox_t *mptr);`
  The mailbox pointed by `mptr` is initialized when `xthread_init_mbox()` is invoked. That is, no message is in the mailbox and the waiting queue in the mailbox is empty.

- `int xthread_send(xthread_mbox_t *mptr, int msg);`
  The function `xthread_send` deposits a message `msg` to the mailbox pointed by `mptr`. If there is a thread waiting for a message, `xthread_send` changes the thread to the ready state and delivers the message to it. On success, `xthread_send` returns 0. On error, it returns -1 if the mailbox is full.

- `int xthread_broadcast(xthread_mbox_t *mptr, int msg);`
  The function `xthread_broadcast` operates much like the function `xthread_send` except that all of the threads waiting in the queue will be unblocked and get the message `msg`.

- `void xthread_recv(xthread_mbox_t *mptr, int *msgptr);`
  The function `xthread_recv` checks whether there is a message in the mailbox pointed by `mptr`. If yes, the message can be delivered to the location pointed by `msgptr`. Otherwise, the calling thread has to wait until a message arrives.



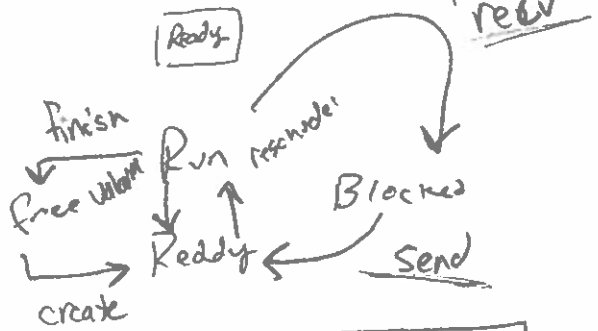
thread #1
foo
recv(&msg)

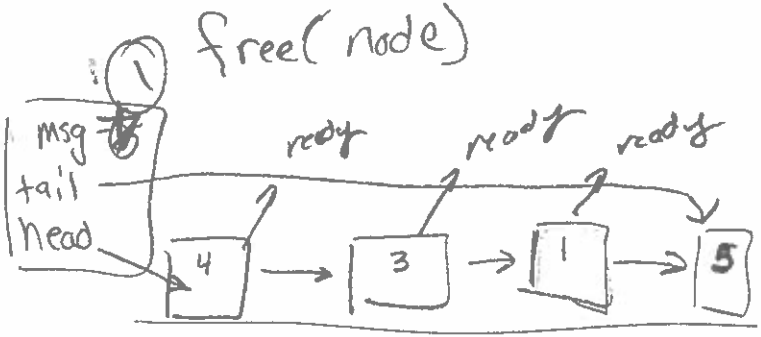
sqrt(4, &msg)

msg = 2

xthread[xid].state = ready

*mptr → head → number = 7





xid

node

recv( )

foo { int msg

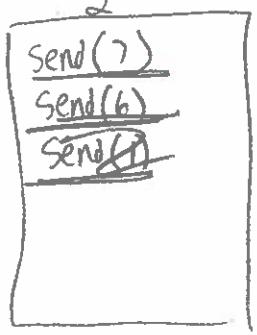
recv(&msg)

print(*msg)

&msg

(7)

recv( )



reschedule(1)

|

reschedule(3)

Send ( )

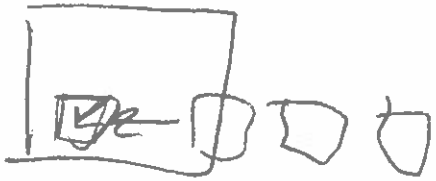
mailbox

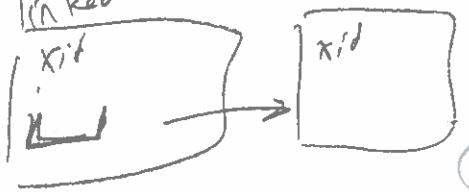
Bar

Send(7)

send

Queue (is Q)





Note that if the SIGALRM occurs during the execution of thread creation/completion, exit/join, etc., the process table may lead to an inconsistent state (why? explain it in your report). To solve this problem, you can use

```
usec = ualarm(0,0);
...
ualarm(usec,0);
```

to disable the timer interrupt at the function entrance and restore it before the function returns.

Turnin

Each group (at most two students) has to submit this project electronically using the following turnin command on grail:

```
turnin -c cis620s -p proj2 Makefile proc.h ctxsw.s create.c message.c ...all...
```

Each group also needs to hand in a hard-copy document which includes the thread state transition diagram, description of your code, experiences in debugging and testing, etc. The cover page should contain your picture(s), name(s) and the login id you used to turnin the project. Start on time and good luck. If you have any questions, send e-mail to `sang@cis.csuohio.edu`.

DIY Threads

user-level → invisible to OS' kernel

```
1  /*      proc.h      */
2  #include <stdio.h>
3  #include <stdlib.h>

4  typedef int WORD;

5  #define PNREGS 5

6  #define NPROC 10

7  #define SP     0

8  /* state */
9  #define XFREE    0
10 #define XREADY   1
11 #define XRUN     2

12 struct xentry  {
13         int  xid;
14         WORD xregs[PNREGS];    /* save SP */
15         WORD xbase;
16         WORD xlimit;
17         int  xstate;
18 };

19 #define STKSIZE 8192

20 extern struct xentry xtab[];

21 extern int currxid;
```

for each thread id

PNREGS: # of register, save register stack pointer register

stack region

one of the states

thread table, defined in page 4 on top

ctxsw(current, next)

context switch function

intel 32-bit assembly: it
- switch from caller thread to another thread → stack changes.

```
1          .file   "ctxsw.s"
2          .version        "01.01"
3  gcc2_compiled.:
4  .text
5          .align 4
6  .globl ctxsw
7          .type    ctxsw,@function
8  ctxsw:
9          pushl %ebp
10         movl  %esp,%ebp          # frame pointer
11         pushl %esi
12         pushl %edi               # preserve registers
13         movl  8(%ebp),%ebx
14         movl  %esp,(%ebx)
15         movl  12(%ebp),%ebx
16         movl  (%ebx),%esp
17         popl  %edi
18         popl  %esi
19 .L1:
20         popl %ebp
21         ret
22 .Lfe1:
23         .size    ctxsw,.Lfe1-ctxsw
24         .ident  "GCC: (GNU) 2.7.2.3"
```

current → %ebp

save esi, edi

next → %ebx

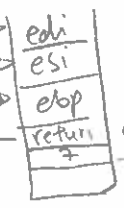
current next

save SP to xreg[sp] i.e. esp, (i.ebx)

of current situation before yield to next thread

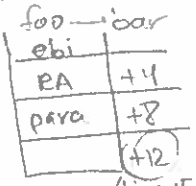
get new stack pointer

change s.p.



add → put ret address foo here to go to foo now

now set (7)



(line 15

file is here

tar xvfz ~cis620s/pub/p1/xt.tar.gz

cd xt

make depend

create.c

```
1  #include <stdio.h>
2  #include <proc.h>

3  /*------------------------------------------------------------------------
4   * userret  --  entered when a thread exits by return
5   *------------------------------------------------------------------------
6   */
7  void userret()
8  {
9     xtab[currxid].xstate = XFREE;
10    printf("XT: Old threads never die; they just fade away. (id:%d)\n",currxid);
11    /* find the next runnable thread to trun */
12    resched();
13 }


14 static int newxid()
15 {
16     int i, xid;
17     static int nextproc =0;
18
19     for(i=0; i<NPROC; i++) { /* find a free process entry */
20         xid = nextproc;
21         if((++nextproc) >= NPROC)
22              nextproc = 0;
23         if(xtab[xid].xstate == XFREE)
24              return(xid);
25     }
26     printf("Error: run out of process table ! \n");
27     exit(1);
28 }


29 /*------------------------------------------------------------------------
30  *  xthread_create  -  create a process to start running a procedure
31  *------------------------------------------------------------------------
32  */
33 int xthread_create(int *procaddr,int nargs, int args)
34 {
35     WORD *saddr;              /* stack address */
36     WORD *ap;
37     struct xentry *xptr;
38     int xid;

39     xid = newxid();

40     xptr = &xtab[xid];
41     xptr->xstate = XREADY;

42     saddr = (WORD *) xptr->xbase;

43     ap = (&args) + nargs;
44     for(; nargs > 0; nargs--)
45         *(--saddr) = *(--ap); /* copy args onto new process' stack */
46     *(--saddr)  = (int)userret;   /* sooner or later you will be there */
47     *(--saddr)  = (int)procaddr;
48     --saddr;            /* for frame ebp; it's not important !? */
49     saddr -= 2;         /* 2 words for si and di */
50     xptr->xregs[SP] = (int) saddr ;

51     return(xid);
52 }
```

ret of stack comes here

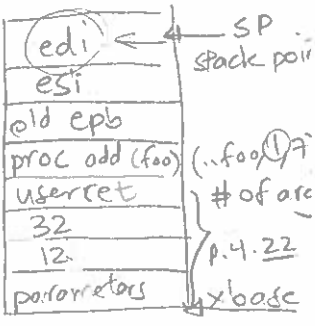
take this stack part and use

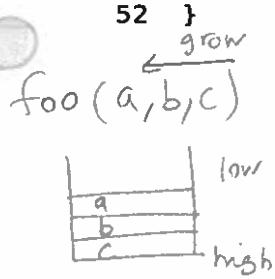
free the level

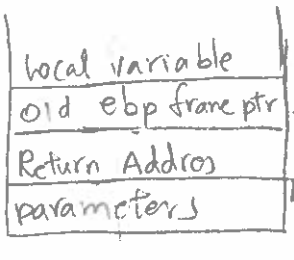
fill the stack with info

save SP to here

recycle fun to recycle freed space







intel 32 bit calling convention.

yield.c

```
1  #include <stdio.h>
2  #include <proc.h>

3  /*------------------------------------------------
4   *  xthread_yield  -  yield control to a thread
5   *------------------------------------------------
6   */
7  void xthread_yield(int xid)
8  {
9      struct xentry *cptr,*xptr;

10     cptr = &xtab[currxid];
11     cptr->xstate = XREADY;
12     xptr = &xtab[xid];
13     xptr->xstate = XRUN;

14     currxid = xid;

15     ctxsw(cptr->xregs,xptr->xregs);
16 }
```

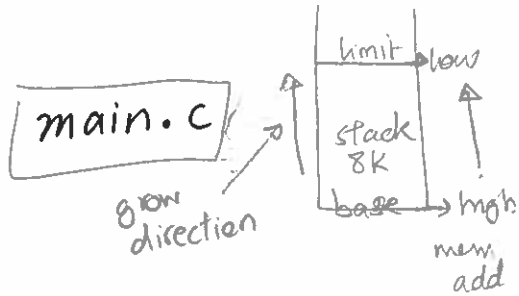
resched.c

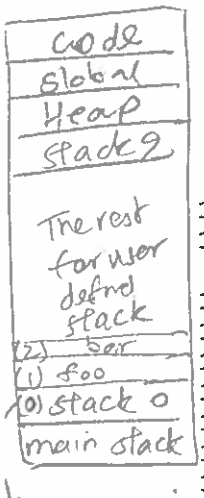
```
1  /* resched.c  -  resched */
2  #include <stdio.h>
3  #include <proc.h>

4  /*------------------------------------------------------------------------
5   * resched  --  find a live thread to run
6   *
7   *------------------------------------------------------------------------
8   */
9  int    resched()
10 {
11     register struct  xentry  *cptr;  /* pointer to old thread entry */
12     register struct  xentry  *xptr;  /* pointer to new thread entry */
13     int i,next;

14     cptr = &xtab[currxid];

15     next = currxid ;
16     for(i=0; i<NPROC; i++) {
17         if( (++next) >= NPROC)
18              next = 0;
19         if(xtab[next].xstate == XREADY) {
20             xtab[next].xstate = XRUN;
21             xptr = &xtab[next];
22             currxid = next;
23             ctxsw(cptr->xregs,xptr->xregs);
24             return;
25         }
26     }
27     printf("XT: no threads to run!\n");
28     exit(0);
29 }
```






```
1   #include <stdio.h>
2   #include <proc.h>

3   extern void xmain();

4   struct xentry xtab[NPROC];
5   int currxid = 0;

6   main(int argc, char *argv[])
7   {
8      register struct xentry *xptr;
9      struct xentry m;
10     int i;
11     int xidxmain;

12     for(i=0 ; i < NPROC; i++){
13        xptr = &xtab[i];
14        xptr->xid = i;
15        xptr->xlimit =  malloc(STKSIZE);
16        xptr->xbase = xptr->xlimit + STKSIZE - sizeof(WORD);
17        xptr->xstate = XFREE;
18     }

19     /* the first thread runs user's xmain with id 0*/
20     xidxmain = xthread_create(xmain, 2, argc, argv);
21     ctxsw(m.xregs, xtab[xidxmain].xregs);
22     /* never be here */
23  }
```

=10 → 10 level stack

xtab

initialize this table xtab[NPROC]

for stack

initialize to empty

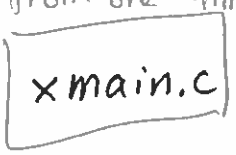
for xmain

create first thread (run xmain)

contact switch

main to xmain

contact switch change stack pointer from one thread to another



xmain.c

user application program

not in the library

```
1   #include <stdio.h>

2   int xidfoo, xidbar;
3   int x=0;

4   int foo(int f)
5   {
6      int i;
7      for(i=0;i<100;i++){
8         printf("This is foo %d, %d\n", f, x++);
9         xthread_yield(xidbar);
10     }
11  }

12  int bar(int p, int q)
13  {
14     int j;
15     for(j=0;j<100;j++){
16        printf("This is bar %d, %d\n", p-q, x++);
17        xthread_yield(xidfoo);
18     }
19  }

20  void xmain(int argc, char* argv[])
21  {
22     xidfoo = xthread_create(foo, 1, 7);
23     xidbar = xthread_create(bar, 2, 32, 12);
24     xthread_yield(xidfoo);
25  }
```

Global

yield yield yield ...

1 input parameter

these 2 parameter are passed

makefile

```
1   #
2   # Makefile for building xthreads library on i386
3   #
4   # Maintain the following definition:
5   #     HFILES      all header files (*.h) programmer created
6   #     SFILES      all C source files (*.c) programmer created
7   #     OFILES      all object files (*.o) required to load program
8   #
9   # Use the following make targets:
10  #     depend      to update header file dependencies

11  .DEFAULT:
12        co -q $@

13  HFILES = proc.h
14  CFILES = create.c yield.c resched.c main.c
15  SFILES = ctxsw.s
16  OFILES = create.o ctxsw.o yield.o resched.o main.o
17  XTLIB  = ./libxt.a
18  APP_CFILES = xmain.c
19  APP_OFILES = xmain.o

20  IFLAGS = -g -I. -I/usr/lib/gcc/x86_64-linux-gnu/7/include
21  CFLAGS =   ${IFLAGS}
22  DEPFLAGS = ${IFLAGS}
23  CC     = gcc -m32
24  AS     = as -32

25  RCS     = Makefile ${HFILES} ${CFILES}

26  a.out: ${XTLIB} ${APP_OFILES}
27        ${CC} ${CFLAGS} ${APP_OFILES} ${XTLIB}

28  ${XTLIB}: ${OFILES}
29        ar cr ${XTLIB} ${OFILES}

30  ci:
31        ci -u ${RCS}

32  clean:
33        rm -f ${OFILES}   ${APP_OFILES}

34  depend:
35        makedepend ${DEPFLAGS} ${CFILES} ${SFILES}
36  # DO NOT DELETE THIS LINE - maketd DEPENDS ON IT
```

ctxsw.o, create.o, yield.o, reschel.o, main.o
put all in libxt.a (line 17)

we are building a package except xmain
xmain is user main which user uses to make his own main function and uses all 4-5 files/package in library → the main P5 function is reserved for package.

```
xmain() {
    xthread_creat()
}
```