George Suarez

David Cruz

CSE 460

Lab 8 - **Dining Philosophers and XV6 Process Priority**

1. **Dining Philosophers and Deadlock**

Try *dine1.cpp* above. Type ^C to check the number of philosophers eating. Run it for some time. What conclusion can you draw on the number of philosophers that can eat at one time? To quit the program, type ^\.

1. **XV6 Process Priority**

* Add *priority* to *struct proc* in *proc.h*:

struct proc {

uint sz;

. . .

char name[16]; // Process name (debugging)

int priority;

. . .

* Assign default priority in **allocproc()** in *proc.c*:

static struct proc\*

allocproc(void)

{

struct proc \*p;

char \*sp;

. . .

found:

p->state = EMBRYO;

p->pid = nextpid++;

p->priority = 10; // default priority

. . .

* Modify **cps()** in *proc.c* discussed in the last lab to include the printout of the priority:

int

cps()

{

struct proc \*p;

// Enable interrupts on this processor.

sti();

//int runningProcesses = 0;

//int sleepingProcesses = 0;

// Loop over process table looking for process with pid.

acquire(&ptable.lock);

cprintf("name \t pid \t state \t\t priority\n");

for (p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if ( p->state == SLEEPING ) {

//sleepingProcesses++;

cprintf("%s \t %d \t SLEEPING \t %d \n ", p->name, p->pid, p->priority);

}

else if ( p->state == RUNNING ) {

//runningProcesses++;

cprintf("%s \t %d \t RUNNING \t %d \n ", p->name, p->pid, p->priority);

}

else if ( p->state == RUNNABLE ) {

cprintf("%s \t %d \t RUNNABLE \t %d\n", p->name, p->pid, p->priority);

}

}

release(&ptable.lock);

return 22;

}

* Modify *foo.c* discussed in Lab 6 so that it loops for a much longer time before exit:

...

for ( z = 0; z < 8000000.0; z += 0.001 )

x = x + 3.14 \* 89.64; // useless calculations to consume CPU time

exit();

...

* Add the function **chpr()** (meaning *change priority*) in *proc.c*:

// change priority

int

chpr( int pid, int priority )

{

struct proc \*p;

acquire(&ptable.lock);

for (p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if (p->pid == pid) {

p->priority = priority;

break;

}

}

release(&ptable.lock);

return pid;

}

* Add **sys\_chpr()** in *sysproc.c*:

int

sys\_chpr ( void )

{

int pid, pr;

if (argint(0, &pid) < 0)

return -1;

if (argint(1, &pr) < 0)

return -1;

return chpr ( pid, pr );

}

* Add **chpr()** as a system call to xv6 as discussed in the last lab.
* Adding *chpr* to *syscall.h:*

…

#define SYS\_close 21

#define SYS\_cps 22

#define SYS\_chpr 23

* Adding *cphr* to *usys.S:*

…

SYSCALL(uptime)

SYSCALL(cps)

SYSCALL(chpr)

* Adding the function prototype for ***cphr***to *syscall.c:*

. . .

extern int sys\_uptime(void);

extern int sys\_cps(void);

extern int sys\_chpr(void);

. . .

. . .

[SYS\_close] sys\_close,

[SYS\_cps] sys\_cps,

[SYS\_chpr] sys\_chpr,

. . .

* Create the user file *nice.c* with which calls **chpr**:

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fcntl.h"

int

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

{

int priority, pid;

if (argc < 3 ) {

printf(2, "Usage: nice pid priority\n");

exit();

}

pid = atoi ( argv[1] );

priority = atoi ( argv[2] );

if ( priority < 0 || priority > 20 ) {

printf(2, "Invalid priority (0-20)!\n");

exit();

}

chpr ( pid, priority );

exit();

}

* Adding *nice* to the *Makefile*

*\_wc\*

*\_nice\*

*\_foo\*

*\_cp\*

*\_ps\*

*...wc.c nice.c cp.c ps.c foo.c zombie.c\*

* Testing the modified **cps()** function using *foo*:

qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512

xv6...

cpu1: starting 1

cpu0: starting 0

sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58

init: starting sh

$ foo 4 &

$ ps

name pid state priority

init 1 SLEEPING 10

sh 2 SLEEPING 10

foo 5 RUNNING 10

foo 4 RUNNABLE 10

foo 6 RUNNABLE 10

foo 7 RUNNABLE 10

foo 8 RUNNABLE 10

ps 9 RUNNING 10

* Changing the priority level of pid 4 from 10 to 18 using the *nice* command

$ nice 4 18

$ ps

name pid state priority

init 1 SLEEPING 10

sh 2 SLEEPING 10

foo 5 RUNNABLE 10

foo 4 RUNNABLE 18

foo 6 RUNNING 10

foo 7 RUNNABLE 10

foo 8 RUNNABLE 10

ps 11 RUNNING 10