Saker Awad - 005263462

Waled Salem - 004893625

Lab8

CSE-460

SCORE: 20/20

1. Dining Philosophers and Deadlock

Try dine1.cpp. Type ^C to check the number of philosophers eating. What conclusion can you draw on the number of philosophers that can eat at one time?

- What this output shows is that it only shows that only one philosopher is allowed to eat at a time.

```
[005263462@csusb.edu@jb359-4 Lab8]$ g++ -o  dine1 dine1.cpp -lSDL
[005263462@csusb.edu@jb359-4 Lab8]$ ./dine1
^C
1 philospers eating
^C
1 philospers eating
^C
1 philospers eating
^C
1 philospers eating
```

- Compile and run dine2.cpp, and repeat the experiment as above. What is the maximum number of philosopers who can eat simultaneously?

- A max of 2 philosophers can eat simultaneously .

```
[005263462@csusb.edu@jb359-4 Lab8]$ g++ -o  dine2 dine2.cpp -lSDL
[005263462@csusb.edu@jb359-4 Lab8]$ ./dine2

Philosoper 2
Taking chopstick 2
Taking chopstick 3
Philosopher 2 eating!

Philosoper 1
Taking chopstick 1
Philosoper 3
Taking chopstick 2
Philosopher 1 eating!

Taking chopstick 3
Taking chopstick 4
Philosopher 3 eating!

Philosoper 0
Taking chopstick 0
Taking chopstick 1
Philosopher 0 eating!

Philosoper 3
Taking chopstick 3
Taking chopstick 4
Philosopher 3 eating!

^CPhilosoper 4
2 philospers eating

Philosoper 0
Taking chopstick 0
Taking chopstick 1
Philosopher 0 eating!

Taking chopstick 4
Philosoper 2
Taking chopstick 2
Taking chopstick 3
Philosopher 2 eating!

^CPhilosoper 1
1 philospers eating

Taking chopstick 1
Taking chopstick 2
Philosopher 1 eating!

Taking chopstick 0
Philosopher 4 eating!

Philosoper 3
Taking chopstick 3
^\Philosoper 2
Quitting, please wait....

Unlocking 0
Unlocking 0 done
Unlocking 1
Unlocking 1 done
Unlocking 2
Unlocking 2 done
Unlocking 3
Unlocking 3 done
Unlocking 4
Taking chopstick 2
Taking chopstick 4
Philosopher 3 eating!

Unlocking 4 done
Taking chopstick 3
Philosopher 2 eating!

Philosoper 0
Taking chopstick 0
Taking chopstick 1
Philosopher 0 eating!
[005263462@csusb.edu@jb359-4 Lab8]$
```

Add a delay statement like SDL_Delay (rand() % 2000); right after the take_chops(l) statement in the philosoper() function. Run the program for a longer time. What do you observe?

Added function:

```
SDL_SemWait ( chopLock[l] );
   take_chops ( l );
   SDL_Delay ( rand() % 2000 );  //could lead to deadlock
   SDL_SemWait ( chopLock[r] );
   take_chops ( r );
```

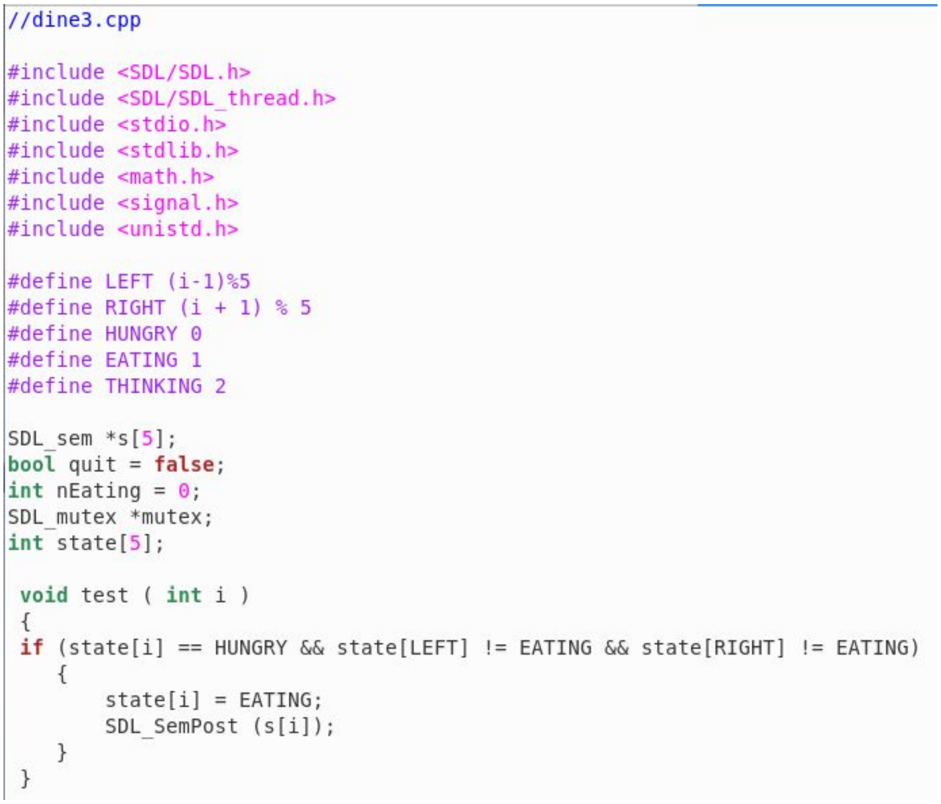
```
[005263462@csusb.edu@jb359-4 Lab8]$ g++ -o  dine2 dine2.cpp -lSDL
[005263462@csusb.edu@jb359-4 Lab8]$ ./dine2

Philosoper 2
Taking chopstick 2
Philosoper 1
Taking chopstick 1
```

```
Philosoper 3
Taking chopstick 3
Philosoper 0
Taking chopstick 0
Taking chopstick 4
Philosopher 3 eating!

Philosoper 4
Taking chopstick 3
Philosopher 2 eating!

Taking chopstick 4
Taking chopstick 2
Philosopher 1 eating!
^C
1 philospers eating

Philosoper 2
Philosoper 3
^CTaking chopstick 3
1 philospers eating
^C
1 philospers eating
^C
1 philospers eating
^C
1 philospers eating

Taking chopstick 2
Taking chopstick 1
Philosopher 0 eating!
^C
1 philospers eating

^\Philosoper 1
Quitting, please wait....

Unlocking 0
Unlocking 0 done
Unlocking 1
Taking chopstick 0
Philosopher 4 eating!
```

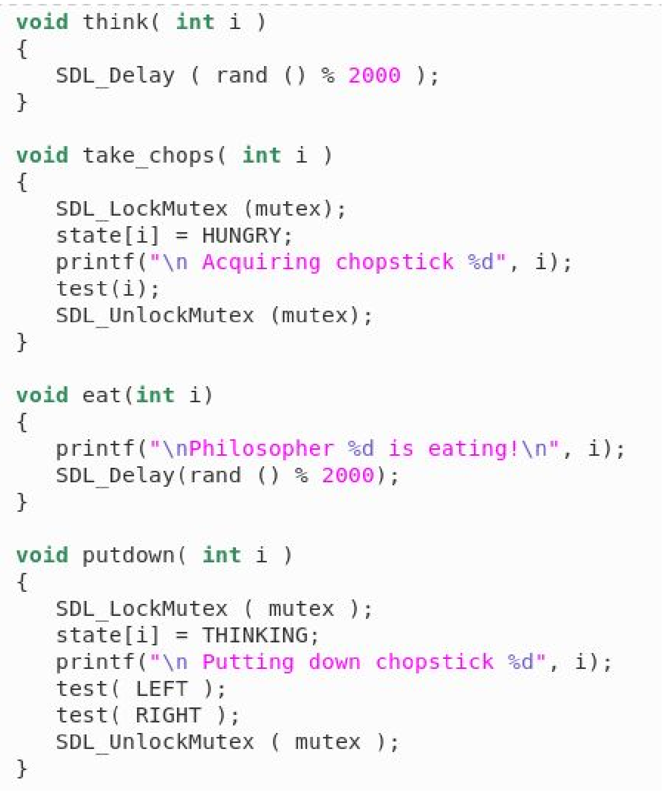
```
Unlocking 1 done
Unlocking 2
Unlocking 2 done
Unlocking 3
Taking chopstick 1
Unlocking 3 done
Unlocking 4
Unlocking 4 done
Taking chopstick 4
Philosopher 3 eating!

Taking chopstick 3
Philosopher 2 eating!

Taking chopstick 2
Philosopher 1 eating!
[005263462@csusb.edu@jb359-4 Lab8]$
```

-Implement this mechanism as discussed in class and call your program dine3.cpp. Repeat the above experiment to see whether deadlock occurs and what the maximum number of philosophers can dine simultaneously.

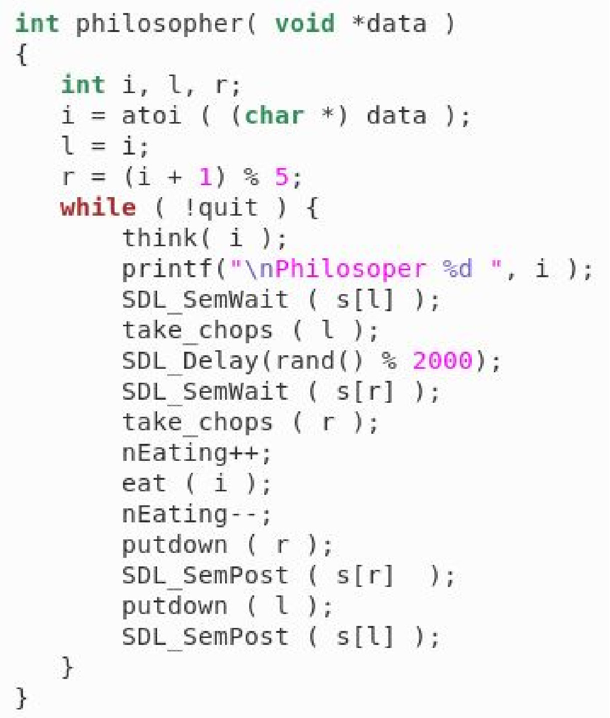
Dine3.cpp code:



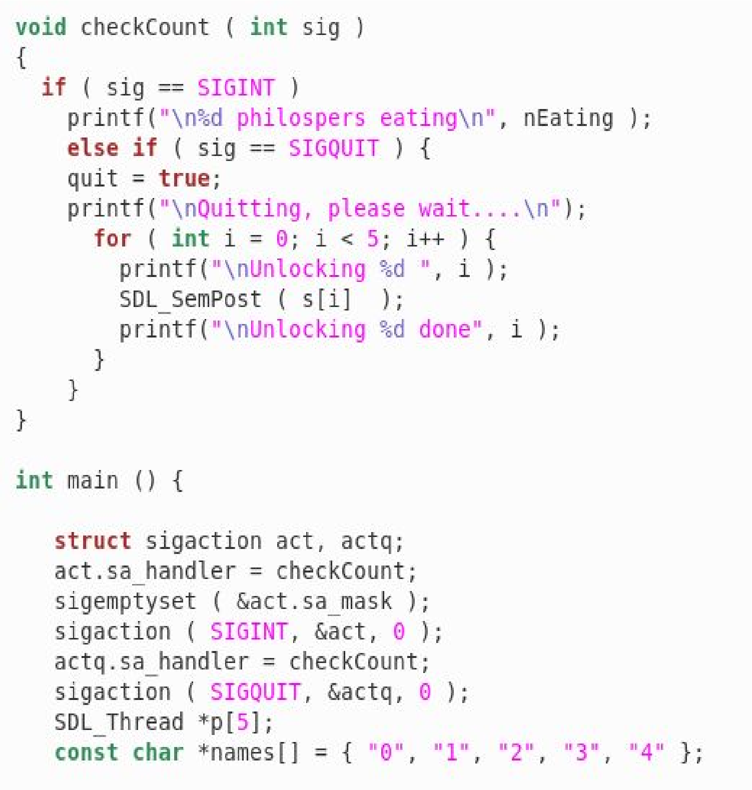
```
//dine3.cpp

#include <SDL/SDL.h>
#include <SDL/SDL_thread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <signal.h>
#include <unistd.h>

#define LEFT (i-1)%5
#define RIGHT (i + 1) % 5
#define HUNGRY 0
#define EATING 1
#define THINKING 2

SDL_sem *s[5];
bool quit = false;
int nEating = 0;
SDL_mutex *mutex;
int state[5];

 void test ( int i )
 {
 if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
    {
        state[i] = EATING;
        SDL_SemPost (s[i]);
    }
 }
```



```
 void think( int i )
 {
    SDL_Delay ( rand () % 2000 );
 }

 void take_chops( int i )
 {
    SDL_LockMutex (mutex);
    state[i] = HUNGRY;
    printf("\n Acquiring chopstick %d", i);
    test(i);
    SDL_UnlockMutex (mutex);
 }

 void eat(int i)
 {
    printf("\nPhilosopher %d is eating!\n", i);
    SDL_Delay(rand () % 2000);
 }

 void putdown( int i )
 {
    SDL_LockMutex ( mutex );
    state[i] = THINKING;
    printf("\n Putting down chopstick %d", i);
    test( LEFT );
    test( RIGHT );
    SDL_UnlockMutex ( mutex );
 }
```



```
int philosopher( void *data )
{
   int i, l, r;
   i = atoi ( (char *) data );
   l = i;
   r = (i + 1) % 5;
   while ( !quit ) {
       think( i );
       printf("\nPhilosoper %d ", i );
       SDL_SemWait ( s[l] );
       take_chops ( l );
       SDL_Delay(rand() % 2000);
       SDL_SemWait ( s[r] );
       take_chops ( r );
       nEating++;
       eat ( i );
       nEating--;
       putdown ( r );
       SDL_SemPost ( s[r]  );
       putdown ( l );
       SDL_SemPost ( s[l] );
   }
}
```



```
void checkCount ( int sig )
{
  if ( sig == SIGINT )
    printf("\n%d philospers eating\n", nEating );
    else if ( sig == SIGQUIT ) {
    quit = true;
    printf("\nQuitting, please wait....\n");
      for ( int i = 0; i < 5; i++ ) {
        printf("\nUnlocking %d ", i );
        SDL_SemPost ( s[i]  );
        printf("\nUnlocking %d done", i );
      }
    }
}

int main () {

   struct sigaction act, actq;
   act.sa_handler = checkCount;
   sigemptyset ( &act.sa_mask );
   sigaction ( SIGINT, &act, 0 );
   actq.sa_handler = checkCount;
   sigaction ( SIGQUIT, &actq, 0 );
   SDL_Thread *p[5];
   const char *names[] = { "0", "1", "2", "3", "4" };
```

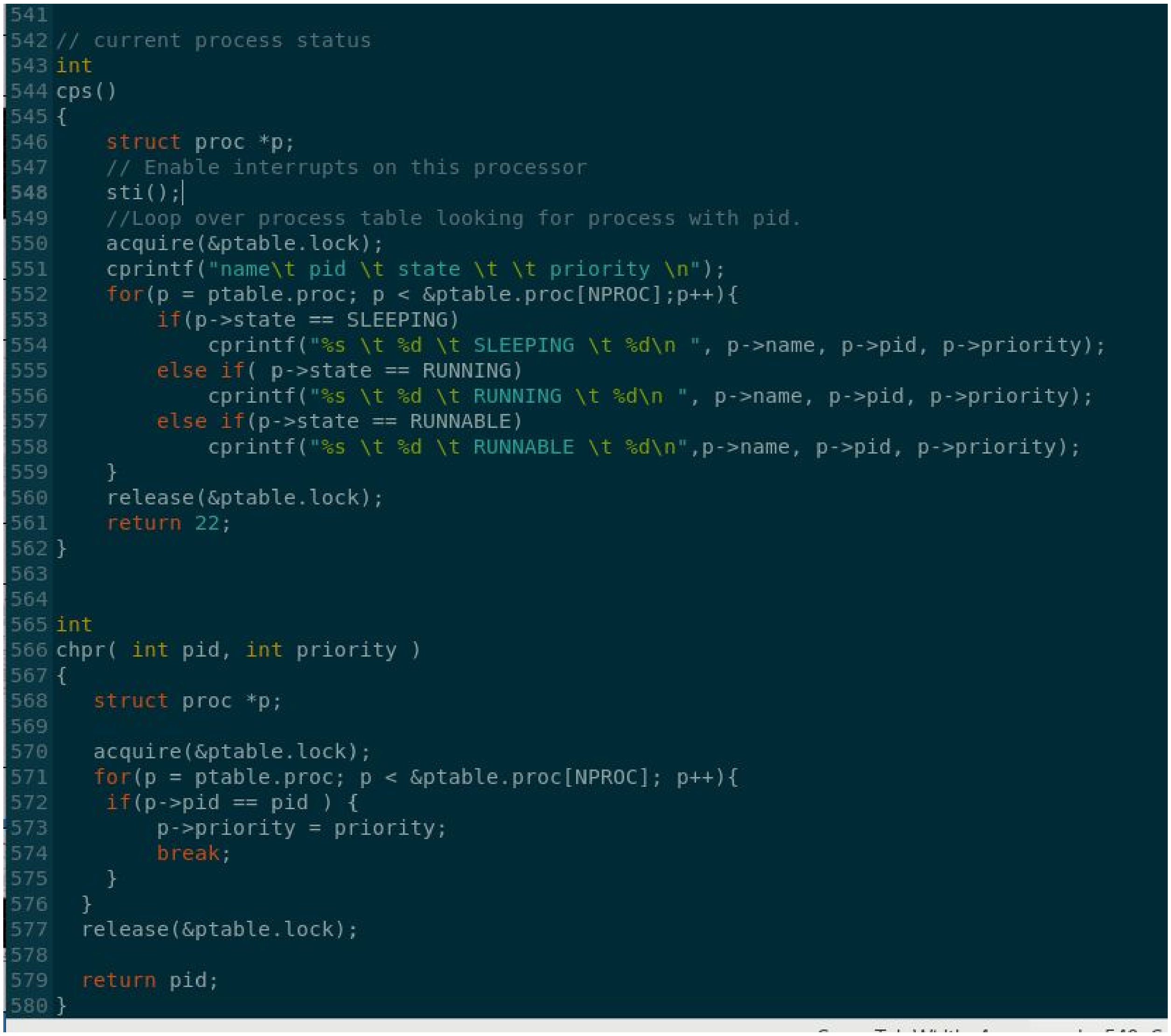
When we run dine3.cpp, 2 philosophers can eat at the same time, with no deadlock. The program uses mutex to make sure that deadlock does not occur & keeps track of each philosopher's state

Dine3.cpp Output :

```
[005263462@csusb.edu@jb359-4 Lab8]$ g++ -o dine3 dine3.cpp -lSDL
[005263462@csusb.edu@jb359-4 Lab8]$ ./dine3

Philosoper 2
Taking choptsticks 2
Philosoper 1
Taking choptsticks 1
Philosoper 3
Taking choptsticks 3
Philosoper 0
Taking choptsticks 0
Taking choptsticks 4
Philosopher 3 eating!

Philosoper 4
Taking choptsticks 2
Philosopher 1 eating!

Taking choptsticks 3
Philosopher 2 eating!

Taking choptsticks 1
Philosopher 0 eating!

Philosoper 1
Taking choptsticks 1
Taking choptsticks 4
Philosoper 3
^CTaking choptsticks 3
0 philospers eating

Taking choptsticks 0
Philosopher 4 eating!

Philosoper 0
Taking choptsticks 0
Philosoper 2
```

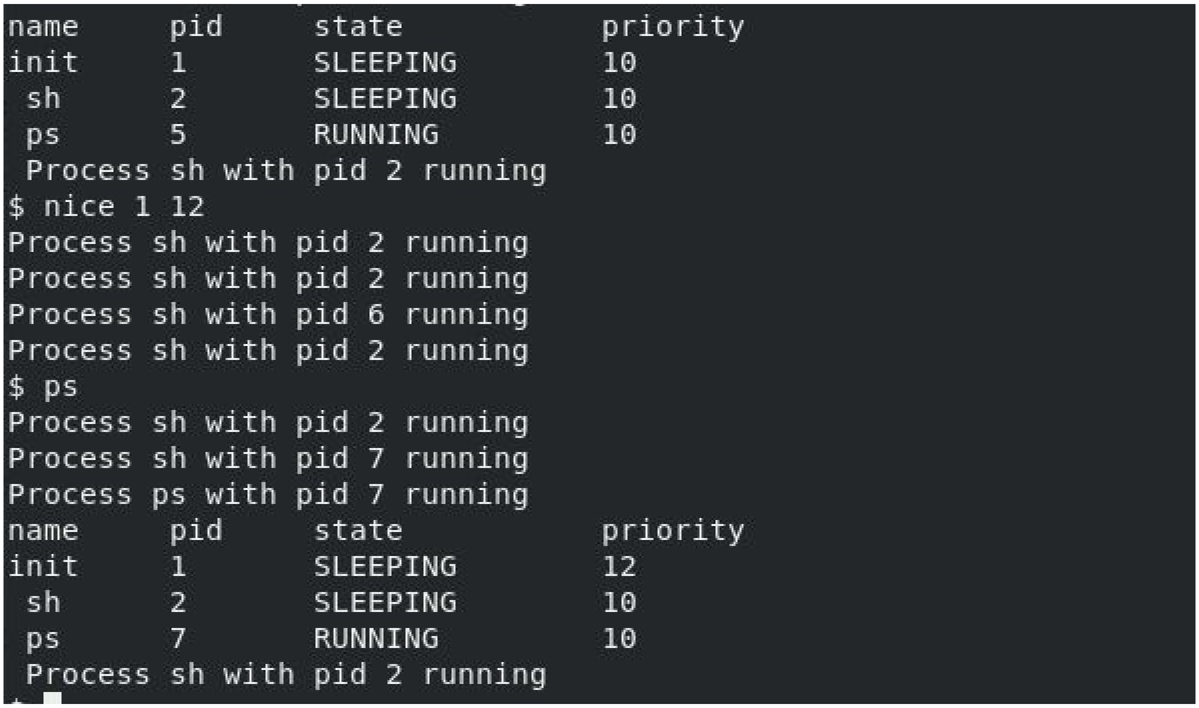
```
Taking choptsticks 2
Taking choptsticks 2
Philosopher 1 eating!
^C
2 philospers eating

Taking choptsticks 3
Philosopher 2 eating!

Taking choptsticks 4
Philosopher 3 eating!
^C
3 philospers eating

Taking choptsticks 1
Philosopher 0 eating!
^C
3 philospers eating

Philosoper 4
Taking choptsticks 4
Philosoper 2
^CTaking choptsticks 2
0 philospers eating

Philosoper 1
Taking choptsticks 1
Philosoper 0
Taking choptsticks 0
Taking choptsticks 1
Philosopher 0 eating!

Taking choptsticks 3
Philosopher 2 eating!
^\
Quitting, please wait....

Unlocking 0
Unlocking 0 done
Unlocking 1
Unlocking 1 done
Unlocking 2
Unlocking 2 done
Unlocking 3
Unlocking 3 done
Unlocking 4
Unlocking 4 done
Philosoper 3
Taking choptsticks 3
Taking choptsticks 2
Philosopher 1 eating!

Taking choptsticks 0
Philosopher 4 eating!

Taking choptsticks 4
Philosopher 3 eating!
[005263462@csusb.edu@jb359-4 Lab8]$
```

2. Source Code



```

// current process status
int
cps()
{
    struct proc *p;
    // Enable interrupts on this processor
    sti();
    //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)
            cprintf("%s \t %d \t SLEEPING \t %d\n ", p->name, p->pid, p->priority);
        else if( p->state == RUNNING)
            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;
}


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;
}
```

Output:



```
name     pid     state           priority
init     1       SLEEPING        10
 sh      2       SLEEPING        10
 ps      5       RUNNING         10
 Process sh with pid 2 running
$ nice 1 12
Process sh with pid 2 running
Process sh with pid 2 running
Process sh with pid 6 running
Process sh with pid 2 running
$ ps
Process sh with pid 2 running
Process sh with pid 7 running
Process ps with pid 7 running
name     pid     state           priority
init     1       SLEEPING        12
 sh      2       SLEEPING        10
 ps      7       RUNNING         10
 Process sh with pid 2 running
$
```