Brian Duenas

CSE 460

# Dining Philosophers and XV6 Process Priority

## 1. Dining Philosophers and Deadlock

**Q:** Try dine1.cpp. 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?

**Output:**

```
[user@csusb.edu@jb359-1 lab8]$ dine1
^C
1 philospers eating
^C
1 philospers eating
^C
1 philospers eating
^C
1 philospers eating
^C
1 philospers eating
^\
Quitting, please wait...
```

**A:** Only one philosopher can eat at a time.

**Q:** Compile and run dine2.cpp, and repeat the experiment as above. What is the maximum number of philosophers who can eat simultaneously?

**Output:**

```
[user@csusb.edu@jb359-1 lab8]$ dine2

Philosoper 2
Taking chopstick 2
Taking chopstick 3
Philosopher 2 eating!
^C
1 philospers eating

Philosoper 1
Taking chopstick 1
Philosoper 3
Taking chopstick 3
Taking chopstick 4
Taking chopstick 2
Philosopher 1 eating!

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!

Philosoper 4
Philosoper 0
Taking chopstick 0
Taking chopstick 1
Philosopher 0 eating!
^C
2 philospers eating

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

Philosoper 1
Taking chopstick 0
Philosopher 4 eating!

Taking chopstick 1
Taking chopstick 2
Philosopher 1 eating!

Philosoper 3
Taking chopstick 3
^ CPhilosoper 2
2 philospers eating

^\
Quitting, please wait....
```

**A:** Only a maximum of two philosophers can eat simultaneously.

**Q:** 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?

**Output:**

```
[user@csusb.edu@jb359-1 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!
```

```
Philosoper 2
Philosoper 3
Taking chopstick 3
Taking chopstick 2
Taking chopstick 1
Philosopher 0 eating!

Taking chopstick 1
Taking chopstick 0
Philosopher 4 eating!

Taking chopstick 4
Philosopher 3 eating!

Philosoper 0
Taking chopstick 0
Taking chopstick 3
Philosopher 2 eating!

Philosoper 4
Taking chopstick 4
Taking chopstick 2
Philosopher 1 eating!

Philosoper 2
Philosoper 3
Taking chopstick 3
Taking chopstick 2
Taking chopstick 1
Philosopher 0 eating!

Taking chopstick 0
Philosopher 4 eating!

Taking chopstick 4
Philosopher 3 eating!

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

Philosoper 3
Philosoper 0
Taking chopstick 0
Taking chopstick 3
Philosoper 2
Taking chopstick 2
Philosoper 4
Taking chopstick 4
^ C
0 philospers eating
^C
0 philospers eating
^C
0 philospers eating
^\
Quitting, please wait....
```

**A:** The philosophers arrive at a deadlock because each has one of the chopsticks.

**Q:** 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.

**Code:**

```
#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 EATING 1
#define THINKING 2

bool quit = false;
SDL_mutex *mutex;
int state[5];
SDL_cond *pickUpCond;

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

void print_info()
{
        int n = 0, a[2];
        SDL_LockMutex(mutex);
        for (int i = 0; i < 5; i++)
                if (state[i] == EATING)
                        a[n++] = i;
        SDL_UnlockMutex(mutex);

        printf("# of philosophers eating is %d: ", n);
        for (int k = 0; k < n; k++) {
                printf("%d", a[k]);
                if (k < n - 1)
                        printf(", ");
        }
        printf("\n");
}

void test(int i)
{
        int left = LEFT;
        if (left < 0) left += 5;

        SDL_LockMutex(mutex);
        while (state[left] == EATING || state[RIGHT] == EATING) {
                SDL_CondWait(pickUpCond, mutex);
        }
        state[i] = EATING;
        SDL_UnlockMutex(mutex);
}
```

```
void putdown(int i)
{
        SDL_LockMutex(mutex);
        state[i] = THINKING;
        SDL_CondBroadcast(pickUpCond);
        SDL_UnlockMutex(mutex);
}

void think(int i)
{
        SDL_Delay(rand() % 3000);
}

int info(void *data)
{
        while (!quit) {
                SDL_Delay(1000);
                print_info();
        }
}
int philosopher(void *data)
{
        int i;
        i = atoi((char *)data);

        while (!quit) {
                think(i);
                test(i);
                eat(i);
                putdown(i);
        }
}

void checkCount(int sig)
{
        if (sig == SIGINT) {
                printf("------------\n");
                print_info();
                printf("------------\n");
        }
        else if (sig == SIGQUIT) {
                quit = true;
                printf("\nQuitting, please wait....\n");
        }
}

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);

        mutex = SDL_CreateMutex();
        if (mutex == NULL) {
                printf("\nMutex creation failed!\n");
                return 1;
        }
        SDL_Thread *p[5];                   //thread identifiers
        const char *names[] = { "0", "1", "2", "3", "4" };
        SDL_Thread *infot;
        pickUpCond = SDL_CreateCond();

        for (int i = 0; i < 5; i++)
                p[i] = SDL_CreateThread(philosopher, (char *)names[i]);

        infot = SDL_CreateThread(info, NULL);

        for (int i = 0; i < 5; i++)
                SDL_WaitThread(p[i], NULL);

        SDL_DestroyMutex(mutex);

        return 0;
}
```

**Output:**

```
[user@csusb.edu@jb359-1 lab8]$ dine3
# of philosophers eating is 1: 2
# of philosophers eating is 1: 4
^ C------------
# of philosophers eating is 1: 4
------------
# of philosophers eating is 1: 4
# of philosophers eating is 2: 2, 4
^ C------------
# of philosophers eating is 2: 2, 4
------------
# of philosophers eating is 2: 2, 4
# of philosophers eating is 2: 1, 4
^ \
Quitting, please wait....
```

**A:** The maximum number of philosophers that can eat at one time is two.

2. XV6 Process Priority

**Q:** Do the experiment as described. Summarize all the steps, including those not presented explicitly above.

**Process:**

1) Add priority to struct proc in proc.h
2) Assign default priority in allocproc() in proc.c
3) Modify cps() in proc.c so it prints out the process priority
4) Modify foo.c so that it loops for a much longer time before exit.
5) Add the function chpr() (meaning change priority) in proc.c
6) Add sys_chpr() in sysproc.c
7) Add name to syscall.h
8) Add function prototype to defs.h
9) Add function prototype to user.h
10) Add function call to sysproc.c
11) Add call to usys.S
12) Add call to syscall.c
13) Create the user file nice.c with which calls chpr

**Output:**

```
$ foo &
$ ps
name     pid     state   priority
init     1       SLEEPING        10
 sh      2       SLEEPING        10
 foo     5       RUNNING         10
 ps      7       RUNNING         10
 Total RUNNING : 2
 Total SLEEPING : 2
$ nice 5 15
$ ps
name     pid     state   priority
init     1       SLEEPING        10
 sh      2       SLEEPING        10
 foo     5       RUNNING         15
 ps      9       RUNNING         10
 Total RUNNING : 2
 Total SLEEPING : 2
$
```