Richard Quintana
Eric Mendez
Cse-460
Lab 9

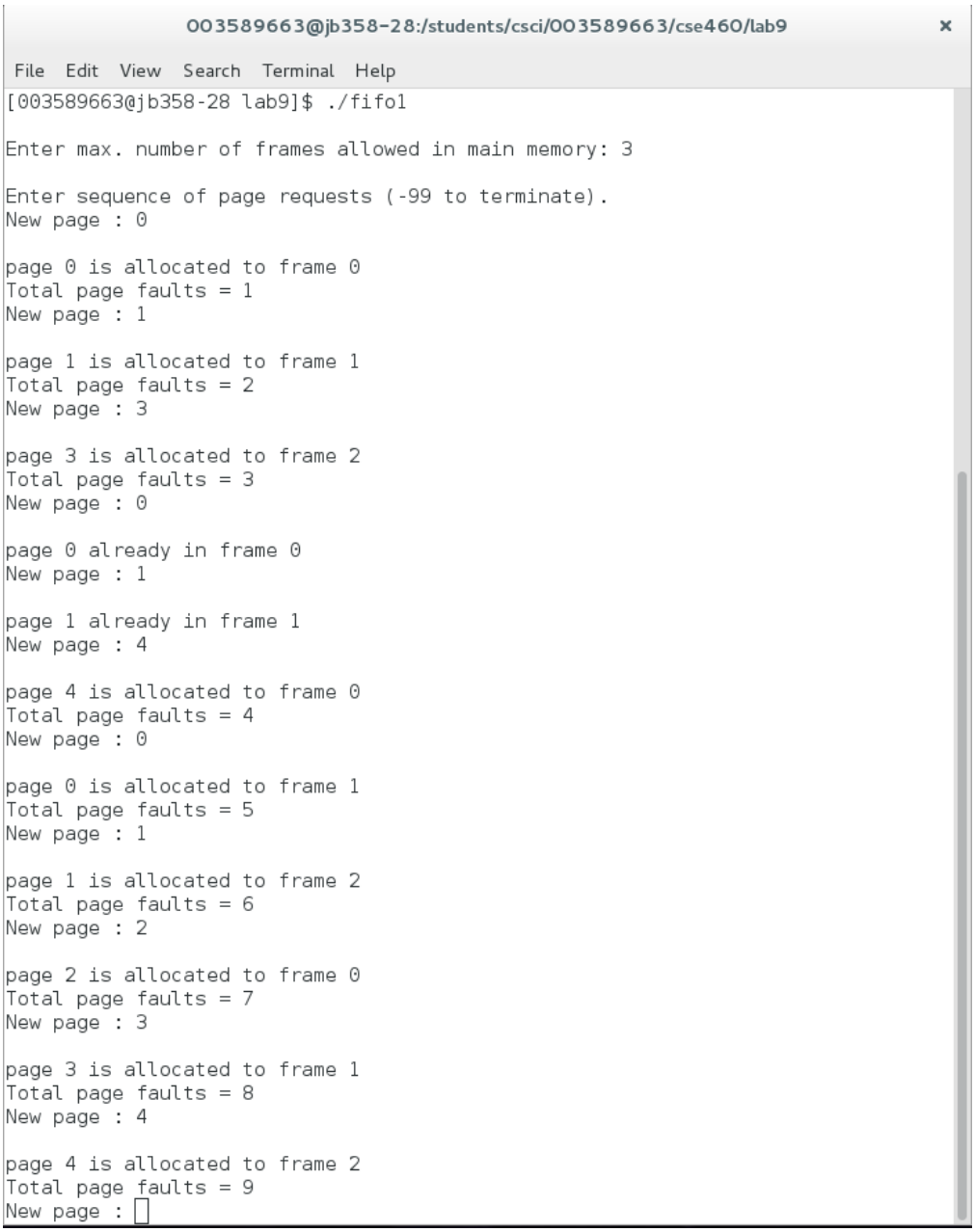
1.**First-in First-out (FIFO) Replacement**

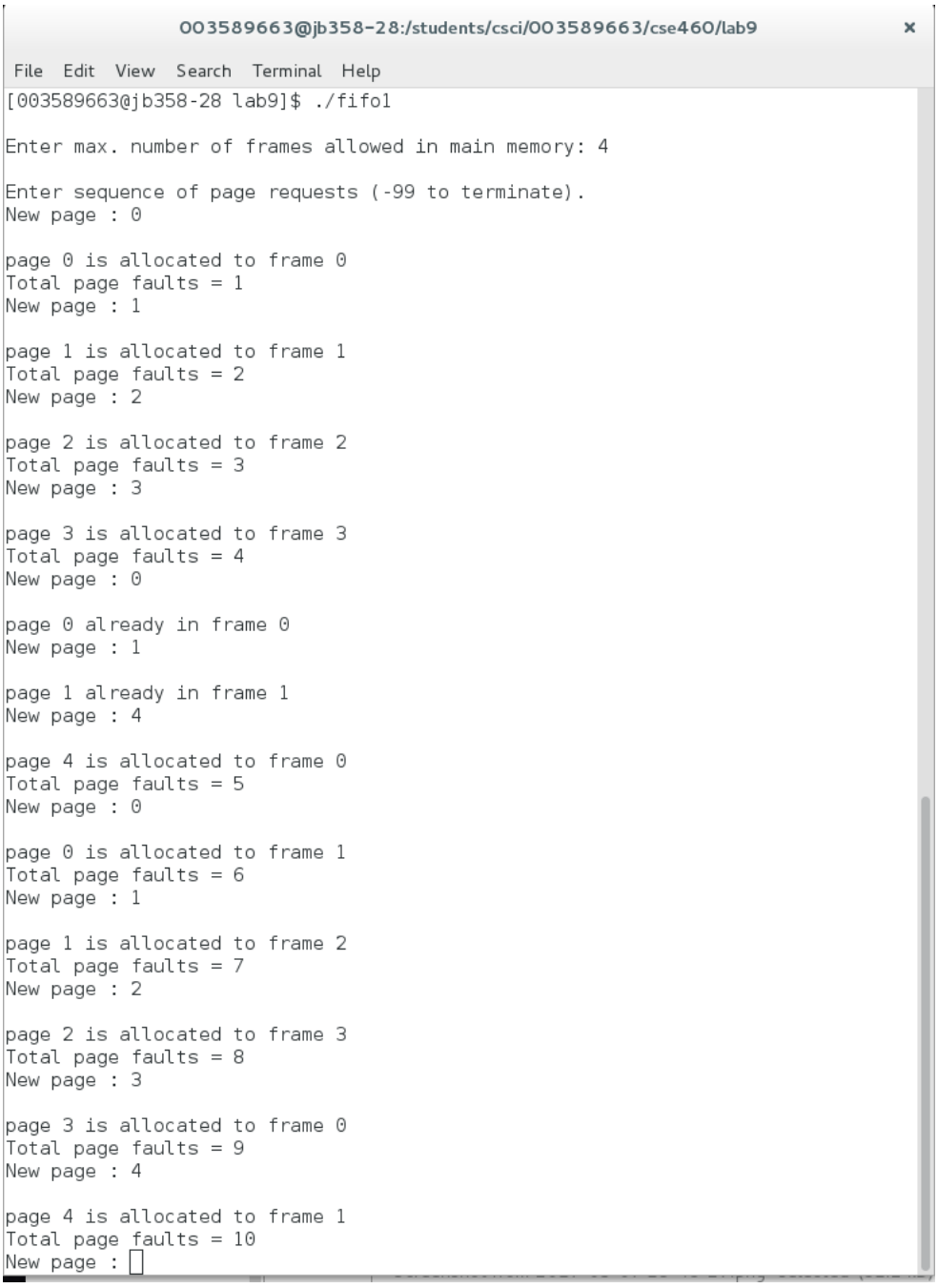
FIFO replacement uses a queue to save the referenced page numbers, replacing the oldest page when the frames are used up. That is, the first one that goes into the queue will be the first one to be removed. Second-chance FIFO is a variation of the straight FIFO algorithm; it includes a reference bit *R*. When it inspects the oldest page, it checks the value of *R*. If *R* = *0*, the page is replaced immediately; if *R* = *1*, it sets *R* to 0 and puts it at the end of the queue and inspects the next oldest page. The following is an implementation of the straight FIFO algorithm, where the *R* bit is not used.

The following is a sample input and output of this program:

```
---------------------------------------------------------------------
   Enter max. number of frames allowed in main memory: 3

   Enter sequence of page requests (-99 to terminate).
   New page : 2

   page 2 is allocated to frame 0
   Total page faults = 1
   New page : 3

   page 3 is allocated to frame 1
   Total page faults = 2
   New page : 2

   page 2 already in frame 0
   New page : -99

   Total number of faults: 2
---------------------------------------------------------------------
```

Compile and execute *fifo1.cpp* listed above. Try the Balady's anomaly examples discussed in class. Did you observe the Belady's anomaly?





Yes Beladys anomaly was observed when the 3 page fram gave 9 fault and the 4 page fault give 10 fault, also specified the specific page reference when there was no page fault

2.Multithreads for FIFO Program

In the above program, the inputs and outputs are mixed and displayed in the same terminal. It would be nice if we separate the inputs and outputs, displaying the inputs in one terminal and the outputs in another one. This can be easily achieved by using the technique we have learned in the past couple of labs. We can first modify *fifo1.cpp* to *fifo2.cpp*, which creates a thread to send the output data to another process using message queues (lab 7). Let us call the program of the receiver process to be *displayMsg.cpp*; it runs and displays data in a different X-terminal from *fifo2*. We can call the thread that sends data in *fifo2.cpp* to be **displayMsg**(), which is similar to the code of *msg2.cpp* of Lab 6. So the program *fifo2.cpp* would look like the following. A condition variable, *updateQueue* is used to synchronize events. The thread sends data only after the **main** function has received new inputs and made the search.

In the code, the statement
sprintf (buffer, "%d,%d,%d\n", page, frame, nFaults);
converts the page number, frame number, and number of faults to text, which is sent to the receiver via a message queue. You can recover these values from the text in the receiver program *displayMsg.cpp* by a statement like the following:
sscanf (some_data.some_text, "%d,%d,%d", &page, &frame, &faults);

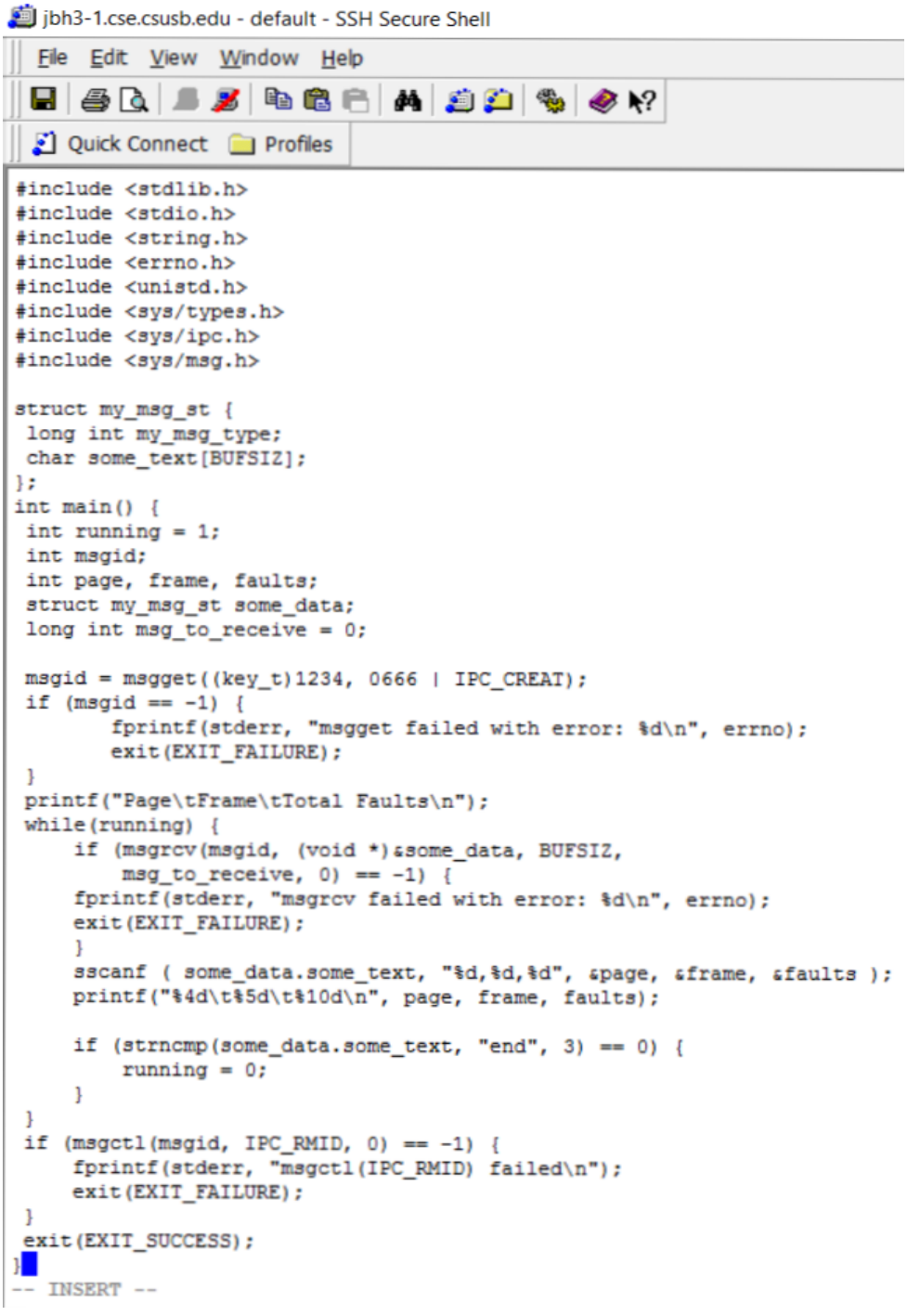
You may display the values with a statement like the following:
printf("%4d\t%5d\t%10d\n", page, frame, faults);

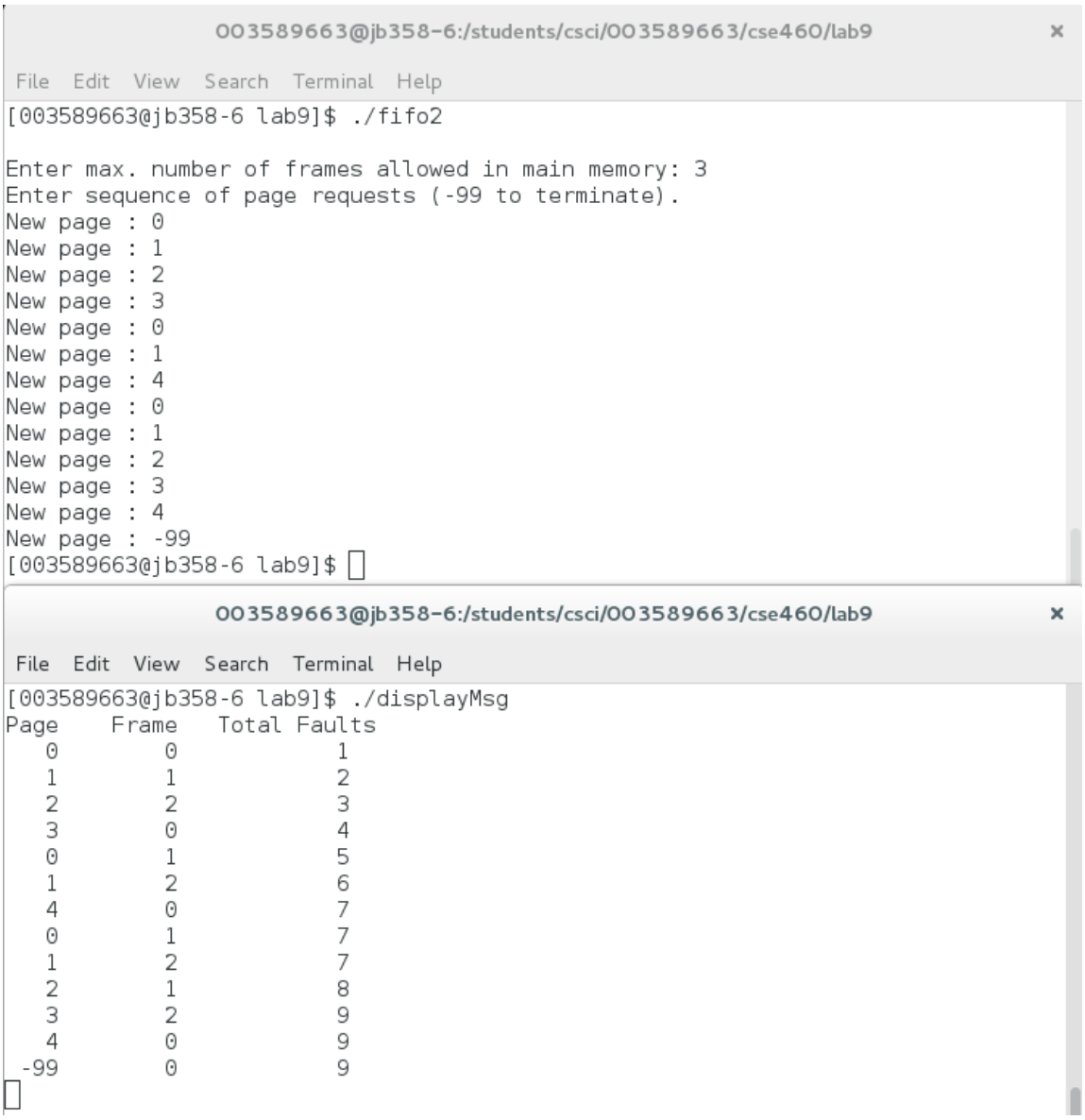
- Implement *displayMsg.cpp*, which is similar to *msg1.cpp* of Lab 7. Run *displayMsg* in one X-term and then *fifo2* in another X-term. Repeat the examples of Belady's anomaly discussed above.

The following are sample segments of displays from the two terminals:

```
$ ./fifo2

Enter max. number of frames allowed in main memory: 3
Enter sequence of page requests (-99 to terminate).
New page : 1
New page : 2
New page : 3
New page : 4
New page : 1
New page : 2
New page : 2
New page : 1
New page : 5
New page : -99
```

```
$ ./displayMsg

Page    Frame   Total Faults
   1        0            1
   2        1            2
   3        2            3
   4        0            4
   1        1            5
   2        2            6
   2        2            6
   1        1            6
   5        0            7
```


jbh3-1.cse.csusb.edu - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct my_msg_st {
long int my_msg_type;
char some_text[BUFSIZ];
};
int main() {
int running = 1;
int msgid;
int page, frame, faults;
struct my_msg_st some_data;
long int msg_to_receive = 0;
msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
if (msgid == -1) {
fprintf(stderr, "msgget failed with error: %d\n", errno);
exit(EXIT_FAILURE);
}
printf("Page\tFrame\tTotal Faults\n");
while(running) {
if (msgrcv(msgid, (void *)&some_data, BUFSIZ,
msg_to_receive, 0) == -1) {
fprintf(stderr, "msgrcv failed with error: %d\n", errno);
exit(EXIT_FAILURE);
}
sscanf (some_data.some_text, "%d,%d,%d", &page, &frame, &faults);
printf("%4d\t%5d\t%10d\n", page, frame, faults);
if (strncmp(some_data.some_text, "end", 3) == 0) {
running = 0;
}
}
if (msgctl(msgid, IPC_RMID, 0) == -1) {
fprintf(stderr, "msgctl(IPC_RMID) failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
-- INSERT --


003589663@jb358-6:/students/csci/003589663/cse460/lab9
File Edit View Search Terminal Help
[003589663@jb358-6 lab9]$ ./fifo2
Enter max. number of frames allowed in main memory: 3
Enter sequence of page requests (-99 to terminate).
New page : 0
New page : 1
New page : 2
New page : 3
New page : 0
New page : 1
New page : 4
New page : 0
New page : 1
New page : 2
New page : 3
New page : 4
New page : -99
[003589663@jb358-6 lab9]$
003589663@jb358-6:/students/csci/003589663/cse460/lab9
File Edit View Search Terminal Help
[003589663@jb358-6 lab9]$ ./displayMsg
Page Frame Total Faults
0 0 1
1 1 2
2 2 3
3 0 4
0 1 5
1 2 6
4 0 7
0 1 7
1 2 7
2 1 8
3 2 9
4 0 9
-99 0 9

```
003589663@jb358-6:/students/csci/003589663/cse460/lab9
File  Edit  View  Search  Terminal  Help
[003589663@jb358-6 lab9]$ ./fifo2

Enter max. number of frames allowed in main memory: 4
Enter sequence of page requests (-99 to terminate).
New page : 0
New page : 1
New page : 2
New page : 3
New page : 0
New page : 1
New page : 4
New page : 0
New page : 1
New page : 2
New page : 3
New page : 4
New page : -99
[003589663@jb358-6 lab9]$
```

```
003589663@jb358-6:/students/csci/003589663/cse460/lab9
File  Edit  View  Search  Terminal  Help
[003589663@jb358-6 lab9]$ ./displayMsg
Page    Frame   Total Faults
   0        0            1
   1        1            2
   2        2            3
   3        3            4
   0        0            4
   1        1            4
   4        0            5
   0        1            6
   1        2            7
   2        3            8
   3        0            9
   4        1           10
 -99        1           10
```

3.Implement One of the Following, *Second Chance* or *LRU*:

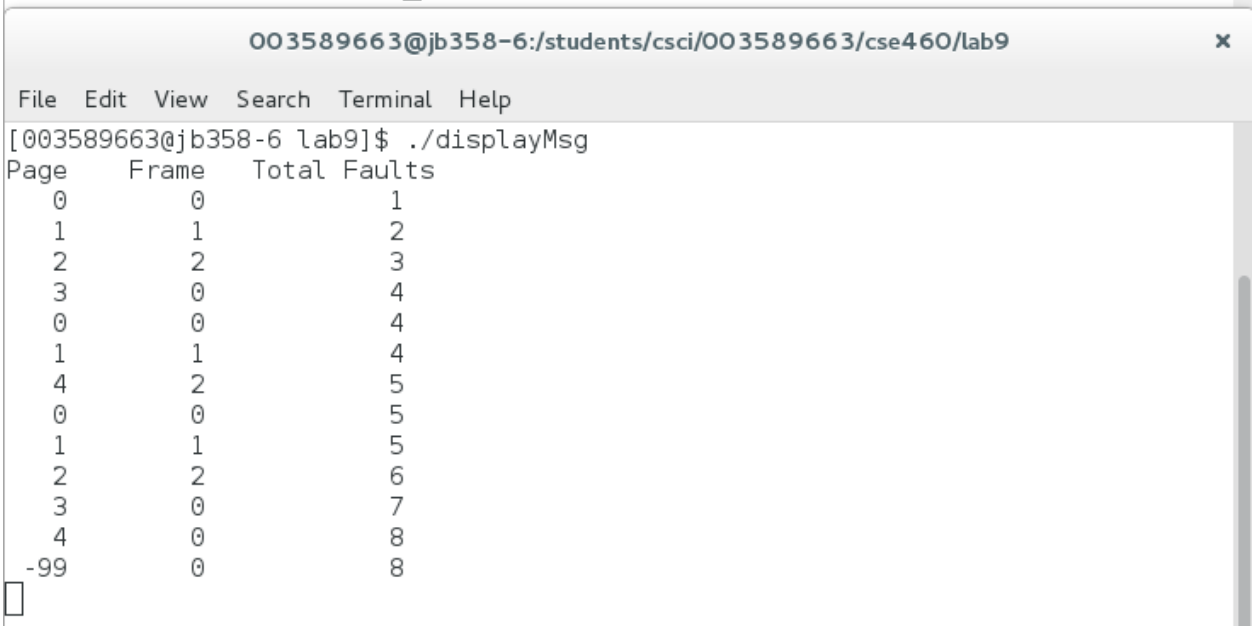
a.Second Chance

Modify *fifo2.cpp* to *fifo3.cpp* to implement the second-chance FIFO replacement discussed above. Compare the total faults for this algorithm and those of *fifo2.cpp*. Which one yields better results.

```
//fifo3.cpp
#include <SDL/SDL.h>
#include <SDL/SDL_thread.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <sys/msg.h>
#include <deque>
#include <errno.h>

using namespace std;

class Cframe {
public:
        int frameNo;
        int pageNo;
        int r;
        Cframe (int n, int p)
 {
        frameNo = n;
        pageNo =  p;
        r = 0;
 }
};
deque <Cframe> Q;
int nFaults = 0;
int page, frame;
SDL_mutex *mutex;
SDL_cond *updateQueue;
bool update = false;
bool quit = false;

#define MAX_TEXT 512

struct my_msg_st {
        long int my_msg_type;
        char some_text[MAX_TEXT];
};

int displayMsg(void *data)
{
        struct my_msg_st some_data;
        int msgid;
        char buffer[BUFSIZ];
        msgid = msgget((key_t)1234, 0666 |
IPC_CREAT);
        if (msgid == -1) {
        fprintf(stderr, "msgget failed with error:
%d\n", errno);
        exit(EXIT_FAILURE);
        }

        while(true) {
        SDL_LockMutex (mutex);
        while(!update && !quit )
        SDL_CondWait (updateQueue, mutex);
        update = false;
        SDL_LockMutex (mutex);
        sprintf(buffer,"%d,%d,%d\n", page, frame,
nFaults );
        some_data.my_msg_type = 1;
        strcpy(some_data.some_text, buffer);

        if(msgsnd(msgid,(void *)&some_data,
MAX_TEXT, 0) == -1) {
        fprintf(stderr, "msgsnd failed\n");
        exit(EXIT_FAILURE);
        }
        if(page == -99)
        break;
        }
        exit(EXIT_SUCCESS);
        }

void fault()
{
  nFaults++;
}

int search(deque<Cframe> &q, int p)
{
        int n = q.size();
        for(int i = 0; i < n; i++ ){
        if(q[i].pageNo == p ) {
        q[i].r = 1;
        return q[i].frameNo;
        }
        }
        return -1;
}

int main()
{
  SDL_Thread *tid = SDL_CreateThread( displayMsg,
(char *) "Send-thread");
```

```
        int maxFrames;
        cout << "\nEnter max. number of frames
allowed in main memory: ";
        cin >> maxFrames;

        int n;
        cout << "Enter sequence of page requests (-
99 to terminate).\n";
        while (true) {
        cout << "New page : ";
        cin >> page;
        if( page == -99) {
        quit = true;
        SDL_CondSignal (updateQueue);
        break;
        }
        if(( frame = search ( Q, page )) != -1) {
        ;
        } else {
        n = Q.size();
        if(n < maxFrames) {
        Cframe aFrame(n, page);
        Q.push_back (aFrame);
        frame = aFrame.frameNo;
        } else {
        int z = 0;
                std::deque<Cframe>::iterator it = Q.begin();
                while(Q[z].r != 0) {
                Q[z].r = 0;
                it++;
                z++;
                if(it == Q.end() ) {
                        it = Q.begin();
                        z = 0;
                }
                }
                Cframe aFrame = Q[z];
                Q.erase(it);
                aFrame.pageNo = page;
                Q.insert (it, aFrame );
                frame = aFrame.frameNo;
                }
                fault();
        }
        SDL_LockMutex (mutex);
        update = true;
        SDL_CondSignal (updateQueue);
        SDL_UnlockMutex (mutex);
        }

        SDL_WaitThread (tid, NULL);
        return 0;
}
```


003589663@jb358-6:/students/csci/003589663/cse460/lab9
File Edit View Search Terminal Help
[003589663@jb358-6 lab9]$ ./fifo3
Enter max. number of frames allowed in main memory: 3
Enter sequence of page requests (-99 to terminate).
New page : 0
New page : 1
New page : 2
New page : 3
New page : 0
New page : 1
New page : 4
New page : 0
New page : 1
New page : 2
New page : 3
New page : 4
New page : -99
[003589663@jb358-6 lab9]$


003589663@jb358-6:/students/csci/003589663/cse460/lab9
File Edit View Search Terminal Help
[003589663@jb358-6 lab9]$ ./displayMsg
Page Frame Total Faults
0 0 1
1 1 2
2 2 3
3 0 4
0 0 4
1 1 4
4 2 5
0 0 5
1 1 5
2 2 6
3 0 7
4 0 8
-99 0 8

```
003589663@jb358-6:/students/csci/003589663/cse460/lab9
File  Edit  View  Search  Terminal  Help
[003589663@jb358-6 lab9]$ ./fifo3

Enter max. number of frames allowed in main memory: 4
Enter sequence of page requests (-99 to terminate).
New page : 0
New page : 1
New page : 2
New page : 3
New page : 0
New page : 1
New page : 4
New page : 0
New page : 1
New page : 2
New page : 3
New page : 4
New page : -99
[003589663@jb358-6 lab9]$
```

```
003589663@jb358-6:/students/csci/003589663/cse460/lab9
File  Edit  View  Search  Terminal  Help
[003589663@jb358-6 lab9]$ ./displayMsg
Page    Frame   Total Faults
   0        0            1
   1        1            2
   2        2            3
   3        3            4
   0        0            4
   1        1            4
   4        2            5
   0        0            5
   1        1            5
   2        2            6
   3        3            6
   4        0            7
 -99        0            7
```

Fifo 3 yields better results.

b.Least-Recently-Used (LRU) Page Replacement

Modify *fifo3.cpp* to *lru.cpp* to implement the LRU replacement using the clock algorithm discussed in class. Compare the total faults for this algorithm and those of *fifo2* and *fifo3*. Which one yields better results?

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <SDL/SDL.h>
#include <SDL/SDL_thread.h>
#include <iostream>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <deque>

using namespace std;

class Cframe {
public:
        int frameNo;
        int pageNo;
        int r;
        Cframe (int n, int p)
 {
        frameNo = n;
        pageNo =  p;
        r = 0;
 }
};
deque <Cframe> Q;
int nFaults = 0;
int page, frame;
SDL_mutex *mutex;
SDL_cond *updateQueue;
bool update = false;
bool quit = false;

#define MAX_TEXT 512

struct my_msg_st {
        long int my_msg_type;
        char some_text[MAX_TEXT];
};

int displayMsg(void *data)
{
        struct my_msg_st some_data;
        int msgid;
        char buffer[BUFSIZ];
        msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
```

```
            if (msgid == -1) {
            fprintf(stderr, "msgget failed with error: %d\n", errno);
            exit(EXIT_FAILURE);
            }

            while(true) {
            SDL_LockMutex (mutex);
            while(!update && !quit )
            SDL_CondWait (updateQueue, mutex);
            update = false;
            SDL_LockMutex (mutex);
            sprintf(buffer,"%d,%d,%d\n", page, frame, nFaults );
            some_data.my_msg_type = 1;
            strcpy(some_data.some_text, buffer);

            if(msgsnd(msgid,(void *)&some_data, MAX_TEXT, 0) == -1) {
            fprintf(stderr, "msgsnd failed\n");
            exit(EXIT_FAILURE);
            }
            if(page == -99)
            break;
            }
            exit(EXIT_SUCCESS);
            }

void fault()
{
  nFaults++;
}

int search(deque<Cframe> &q, int p)
{
            int n = q.size();
            for(int i = 0; i < n; i++ ){
            if(q[i].pageNo == p ) {
            q[i].r = 1;
            return q[i].frameNo;
            }
            }
            return -1;
}

int main()
{
  SDL_Thread *tid = SDL_CreateThread( displayMsg, (char *) "Send-thread");

            int maxFrames;
            cout << "\nEnter max. number of frames allowed in main memory: ";
            cin >> maxFrames;

            int n;
            cout << "Enter sequence of page requests (-99 to terminate).\n";
            while (true) {
            cout << "New page : ";
```

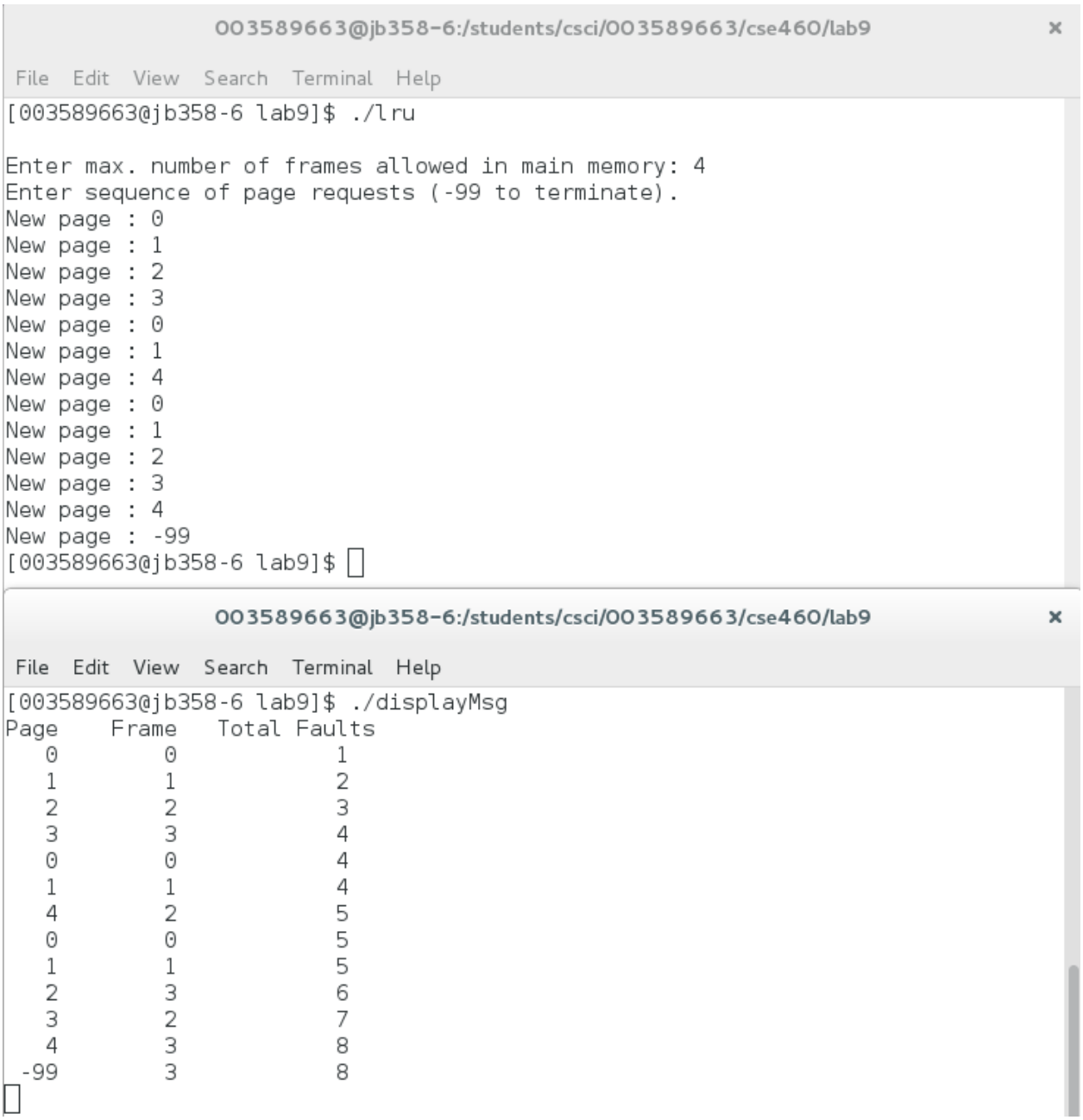
```
            cin >> page;
            if( page == -99) {
            quit = true;
            SDL_CondSignal (updateQueue);
            break;
            }
            if(( frame = search ( Q, page )) != -1) {
            ;
            } else {
            n = Q.size();
            if(n < maxFrames) {
            Cframe aFrame(n, page);
            Q.push_back (aFrame);
            frame = aFrame.frameNo;
            } else {
  while(Q.front().r==1){
  Q.front().r = 0;
  Q.push_back(Q.front());
  Q.pop_front();
  }
  Cframe aFrame = Q.front();
  Q.pop_front();
  aFrame.pageNo = page;
  Q.push_back ( aFrame );
  frame = aFrame.frameNo;
  }
            fault();
  }
  SDL_LockMutex (mutex);
  update = true;
  SDL_CondSignal (updateQueue);
  SDL_UnlockMutex (mutex);
  }

  SDL_WaitThread (tid, NULL);
  return 0;
}
```

```
[003589663@jb358-6 lab9]$ gedit lru.cpp
[003589663@jb358-6 lab9]$ g++ -o lru lru.cpp -lSDL
[003589663@jb358-6 lab9]$ ./lru

Enter max. number of frames allowed in main memory: 3
Enter sequence of page requests (-99 to terminate).
New page : 0
New page : 1
New page : 2
New page : 3
New page : 0
New page : 1
New page : 4
New page : 0
New page : 1
New page : 2
New page : 3
New page : 4
New page : -99
[003589663@jb358-6 lab9]$
```

003589663@jb358-6:/students/csci/003589663/cse460/lab9

File Edit View Search Terminal Help

```
[003589663@jb358-6 lab9]$ ./displayMsg
Page    Frame   Total Faults
   0        0            1
   1        1            2
   2        2            3
   3        0            4
   0        1            5
   1        2            6
   4        0            7
   0        1            7
   1        2            7
   2        0            8
   3        1            9
   4        2           10
 -99        2           10
```



When comparing the toatal faults of the alogorithms of fifo3.cpp lru.cpp fifo2 and fifo3, fifo3 is the program that yields the better results.

4.XV6 Process Priority

In the previous lab, we have learned how to change the priority of a process. In this lab, we will implement a very simple priority scheduling policy. We simply choose a *runnable* process with the highest priority to run. (In practice, multilevel queues are often used to put processes into groups with similar priorities.) As we have done in the previous lab, we assume that a process has a value between 0 and 20, the smaller the value, the higher the priority. The default value is 10. The program *nice* that we implemented in the previous lab is used to change the priority of a process

1.Give high priority to a newly loaded process by adding a *priority* statement in *exec.c*:

```
int
exec(char *path, char **argv)
{
  char *s, *last;
  .....
  proc->tf->esp = sp;
  proc->priority = 2;    // Added statement
  switchuvm(proc);
  freevm(oldpgdir);
  .....
}
```

2.Modify *foo.c* so that the parent waits for the children and adjust the loop for your convenience of observations :

```
 int
main(int argc, char *argv[])
{  .....
  for ( k = 0; k < n; k++ ) {
    id = fork ();
    if ( id < 0 ) {
      printf(1, "%d failed in fork!\n", getpid() );
    } else if ( id > 0 ) {  //parent
      printf(1, "Parent %d creating child  %d\n", getpid(), id );
      wait ();
   } else {   // child
      printf(1, "Child %d created\n",getpid() );
      for ( z = 0; z < 8000000.0; z += 0.01 )
         x =  x + 3.14 * 89.64;   // useless calculations to consume CPU time
      break;
    } }
  exit();
}
```

3.Observe the default round-robin (RR) scheduling.

Round-robin (RR) is the default scheduling algorithm used by xv6. You can see how this works by creating a few *foo* processes in the background and running *ps* a few times at random time intervals in xv6:

```
$ foo &; foo &; foo &

$ ps
name    pid     state           priority
init    1       SLEEPING        2
 sh     2       SLEEPING        2
 foo    9       RUNNING         10
 foo    8       SLEEPING        2
 foo    5       SLEEPING        2
 foo    7       SLEEPING        2
 foo    10      RUNNABLE        10
 foo    11      RUNNABLE        10
 ps     13      RUNNING         2
$ ps
```

```
name    pid     state           priority
init    1       SLEEPING        2
 sh     2       SLEEPING        2
 foo    9       RUNNABLE        10
 foo    8       SLEEPING        2
 foo    5       SLEEPING        2
 foo    7       SLEEPING        2
 foo    10      RUNNING         10
 foo    11      RUNNABLE        10
 ps     14      RUNNING         2

.....
```

You can see that the three *foo* child processes are run alternately while the parents are sleeping.

4.Implement Priority Scheduling.

We can modify the **scheduler** function in *proc.c* to select the highest priority runnable process:

```
#define NULL 0
void
scheduler(void)
{
  struct proc *p;
  struct proc *p1;
  for(;;){
    sti();
    struct proc *highP = NULL;
    // Looking for runnable process
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;
      highP = p;
      // choose one with highest priority
      for(p1 = p + 1; p1 < &ptable.proc[NPROC]; p1++){
        if(p1->state != RUNNABLE)
          continue;
        if ( highP->priority > p1->priority )   // larger value, lower priority
          highP = p1;
      }
      p = highP;
      proc = p;
      switchuvm(p);
      .....
    }
    release(&ptable.lock);
  }
}
```

5.Observe the priority scheduling.

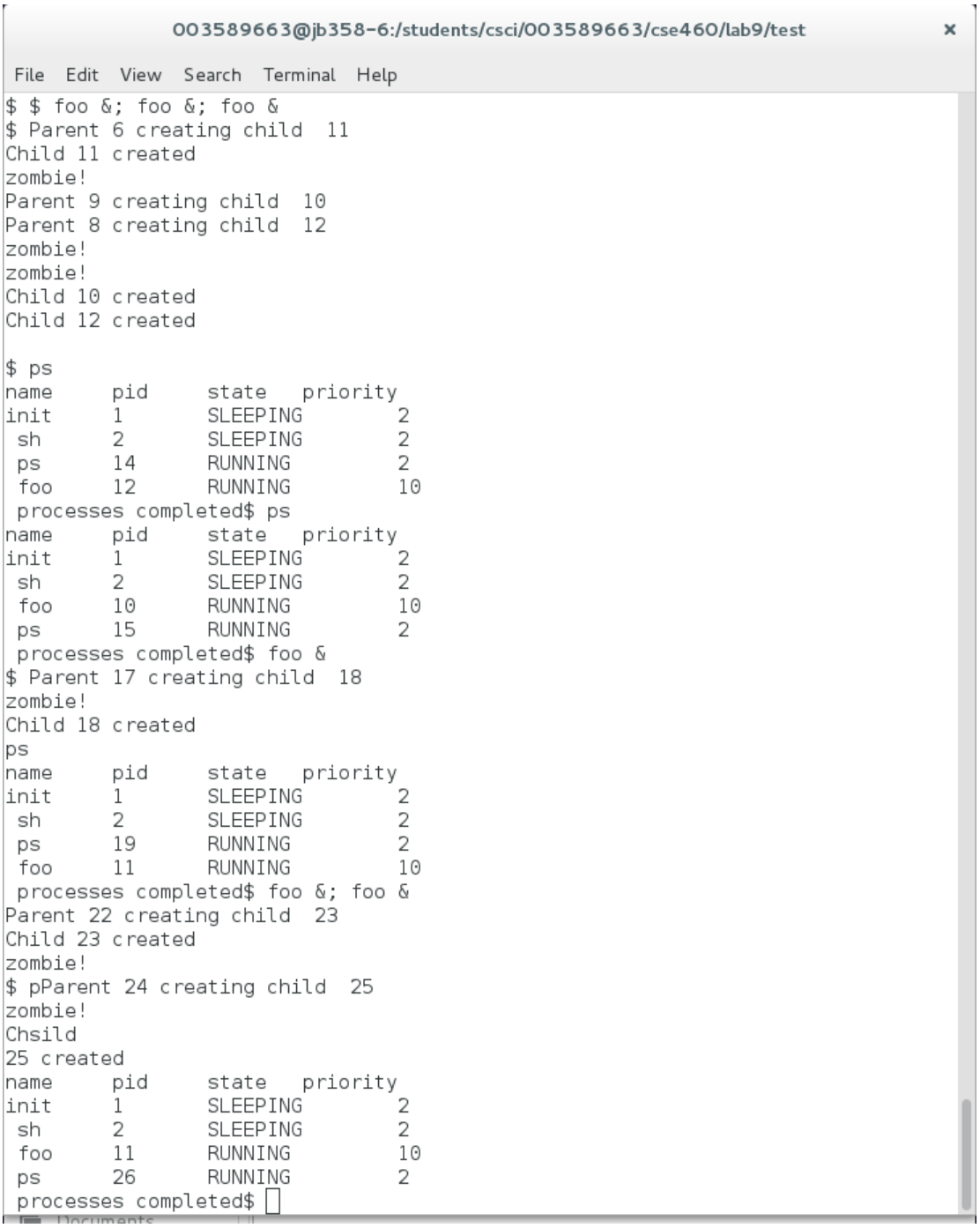
We run xv6 with the scheduler and again use *foo* and *ps* to see how it works. We use *nice* to change the priority of a process.

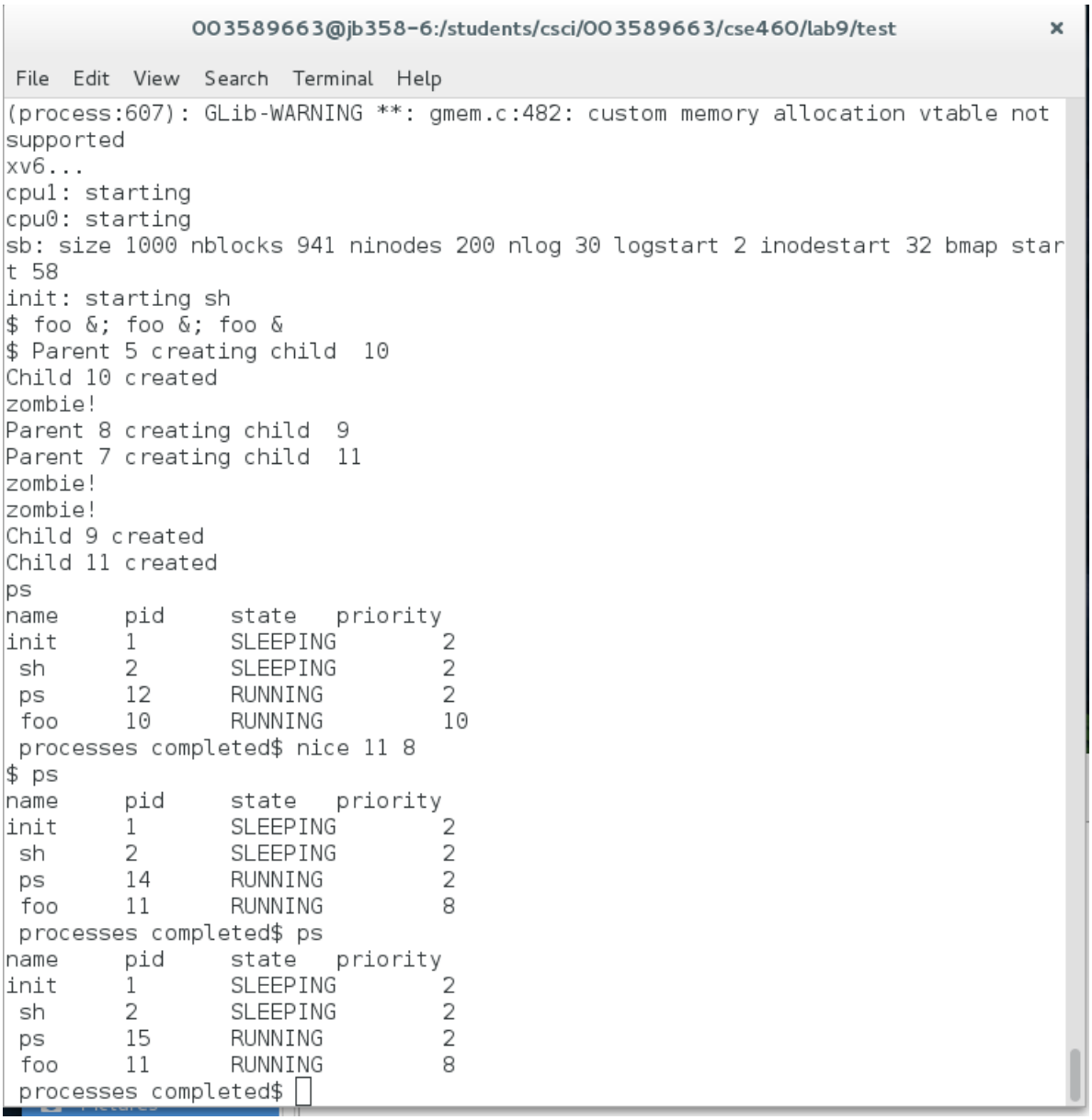
```
$ foo &; foo &; foo &

$ ps
name    pid     state           priority
init    1       SLEEPING        2
 sh     2       SLEEPING        2
 ps     13      RUNNING         2
 foo    10      SLEEPING        2
 foo    5       SLEEPING        2
 foo    7       RUNNING         10
 foo    8       SLEEPING        2
 foo    9       RUNNABLE        10
 foo    11      RUNNABLE        10

$ nice 11 8

$ ps
name    pid     state           priority
init    1       SLEEPING        2
 sh     2       SLEEPING        2

 .....
 ps     15      RUNNING         2
 foo    10      SLEEPING        2
 foo    5       SLEEPING        2
 foo    7       RUNNABLE        10
 foo    8       SLEEPING        2
 foo    9       RUNNABLE        10
 foo    11      RUNNING         8

$ ps
name    pid     state           priority
init    1       SLEEPING        2
 sh     2       SLEEPING        2
 ps     16      RUNNING         2
 foo    10      SLEEPING        2
 foo    5       SLEEPING        2
 foo    7       RUNNABLE        10
 foo    8       SLEEPING        2
 foo    9       RUNNABLE        10
 foo    11      RUNNING         8
```

We can see that after we have changed the priority of process *11* to 8, which is higher than the priority 10 of processes *7* and *9*, process *11* is always selected to run.

Work to do

Do the experiment as described above. Copy-and-paste your outputs and commands to your report. Summarize all the steps, including those not presented explicitly above.


003589663@jb358-6:/students/csci/003589663/cse460/lab9/test
File Edit View Search Terminal Help
$ $ foo &; foo &; foo &
$ Parent 6 creating child 11
Child 11 created
zombie!
Parent 9 creating child 10
Parent 8 creating child 12
zombie!
zombie!
Child 10 created
Child 12 created
$ ps
name pid state priority
init 1 SLEEPING 2
sh 2 SLEEPING 2
ps 14 RUNNING 2
foo 12 RUNNING 10
processes completed$ ps
name pid state priority
init 1 SLEEPING 2
sh 2 SLEEPING 2
foo 10 RUNNING 10
ps 15 RUNNING 2
processes completed$ foo &
$ Parent 17 creating child 18
zombie!
Child 18 created
ps
name pid state priority
init 1 SLEEPING 2
sh 2 SLEEPING 2
ps 19 RUNNING 2
foo 11 RUNNING 10
processes completed$ foo &; foo &
Parent 22 creating child 23
Child 23 created
zombie!
$ pParent 24 creating child 25
zombie!
Chsild
25 created
name pid state priority
init 1 SLEEPING 2
sh 2 SLEEPING 2
foo 11 RUNNING 10
ps 26 RUNNING 2
processes completed$



The experiment was completed successfully with each step as directed.

20/20