George Suarez

David Cruz

CSE 460

Lab 9 – Page Replacement Algorithms and XV6 Priority Scheduling

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

* Compile and Execute *fifo1.cpp*

 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

-------------------------------------------------------------------

* Try the Belady’s anomaly examples discussed in class. Did you observe the Belady’s anomaly?

Yes. When the all the frames are used up, the oldest page has been replaced with a newer page. Also, the number of faults depends on what the maximum of frames has been created, and if a page is already in a certain frame then the page fault is not incremented at all.

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 9 on master\***

**$** ./fifo1

Enter max. number of frames allowed in main memory: 4

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

New page : 0

page 0 is allocated to frame 0

Total page faults = 1

New page : 1

page 1 is allocated to frame 1

Total page faults = 2

New page : 2

page 2 is allocated to frame 2

Total page faults = 3

New page : 3

page 3 is allocated to frame 3

Total page faults = 4

New page : 0

page 0 already in frame 0

New page : 1

page 1 already in frame 1

New page : 4

page 4 is allocated to frame 0

Total page faults = 5

New page : 0

page 0 is allocated to frame 1

Total page faults = 6

New page : 1

page 1 is allocated to frame 2

Total page faults = 7

New page : 2

page 2 is allocated to frame 3

Total page faults = 8

New page : 3

page 3 is allocated to frame 0

Total page faults = 9

New page : 4

page 4 is allocated to frame 1

Total page faults = 10

New page : -99

Total number of faults: 10

1. **Multithreads for FIFO Program**

• Implement displayMsg.cpp. Run displayMsg in one X-term and then fifo2 in another X-term. Repeat the examples of Belady’s anomaly discussed above.

***Fifo2.cpp***

#include <SDL2/SDL.h>

#include <SDL2/SDL\_thread.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <sys/types.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <signal.h>

#include <unistd.h>

#include <iostream>

#include <deque>

using namespace std;

class Cframe

{

public:

int frameNo; // frame number

int pageNo; // page number

int r; // reference bit

Cframe(int n, int p) // constructor

{

frameNo = n;

pageNo = p; // no page loaded at beginning

r = 0;

}

};

deque <Cframe> Q;

int nFaults = 0;

int page, frame;

SDL\_mutex \*mutex1;

SDL\_cond \*updateQueue; //condition variable

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(mutex1);

while (!update && !quit)

SDL\_CondWait(updateQueue, mutex1);

update = false;

SDL\_LockMutex(mutex1);

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(const deque<Cframe> &q, int p)

{

int n = q.size();

for (int i = 0; i < n; i++)

{

if (q[i].pageNo == p)

return q[i].frameNo;

}

return -1;

}

int main()

{

SDL\_Thread \*tid = SDL\_CreateThread(displayMsg, "Send Thread" ,(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

{

Cframe aFrame = Q.front();

Q.pop\_front();

aFrame.pageNo = page;

Q.push\_back(aFrame);

frame = aFrame.frameNo;

}

fault();

}

SDL\_LockMutex(mutex1);

update = true;

SDL\_CondSignal(updateQueue);

SDL\_UnlockMutex(mutex1);

}

SDL\_WaitThread(tid, NULL);

return 0;

}

***displayMsg.cpp***

include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <sys/types.h>

#include <unistd.h>

#include <iostream>

#define MAX\_TEXT 512

struct my\_msg\_st

{

long int my\_msg\_type;

char some\_text[MAX\_TEXT];

};

int main()

{

int running = 1;

struct my\_msg\_st some\_data;

int msgid, page, frame, faults;

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 (1)

{

if (msgrcv(msgid, (void \*)&some\_data, MAX\_TEXT, 0, 0) == -1)

{

fprintf(stderr, "msgrcv failed with error: %d\n", errno);

exit(EXIT\_FAILURE);

}

sscanf(some\_data.some\_text, "%d,%d,%d\n", &page, &frame, &faults);

if (strncmp(some\_data.some\_text, "-99", 3) == 0)

{

printf("\nTerminal ending...\n");

running = 0;

break;

}

printf("%d\t%d\t%d\n", page, frame, faults);

}

if (msgctl(msgid, IPC\_RMID, 0) == -1)

{

fprintf(stderr, "msgctl failed with error: %d\n", errno);

exit(EXIT\_FAILURE);

}

exit(EXIT\_SUCCESS);

}

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 9 on master\***

**$** ./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 : 4

New page : 3

New page : 2

New page : 1

New page : 2

New page : 3

New page : 4

New page : -99

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 9 on master\***

**$** ./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

4 0 7

3 1 8

2 2 9

1 0 10

2 2 10

3 1 10

4 1 11

Terminal ending...

1. **Implement One of the following, Second Chance or LRU:**
2. 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?

fifo2 produces slightly better results since a total of 11 faults were created as opposed to the 12 faults created in fifo3.

***Fifo3.cpp***

#include <iostream>

#include <SDL2/SDL.h>

#include <SDL2/SDL\_thread.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/msg.h>

#include <unistd.h>

#include <deque>

using namespace std;

class Cframe

{

public:

int frameNo;

// frame number

int pageNo;

// page number

int r;

// reference bit

Cframe(int n, int p)

{

frameNo = n;

pageNo = p;

// no page loaded at beginning

r = 0;

}

};

deque<Cframe> Q;

int nFaults = 0;

int page, frame;

SDL\_mutex \*mutex1;

SDL\_cond \*updateQueue; // condition varaible

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(mutex1);

while (!update && !quit)

{

SDL\_CondWait(updateQueue, mutex1);

}

update = false;

SDL\_LockMutex(mutex1);

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;

// recently referenced

return q[i].frameNo;

}

}

return -1;

}

int main()

{

SDL\_Thread \*tid = SDL\_CreateThread(displayMsg, "Send Message" ,(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

{

Cframe aFrame = Q.front();

while (aFrame.r == 1)

{

// find oldest page that

Q.pop\_front();

// has r == 0; set all r

aFrame.r = 0;

// flags to 0 until one is

Q.push\_back(aFrame);

// found

aFrame = Q.front();

}

Q.pop\_front();

aFrame.pageNo = page;

Q.push\_back(aFrame);

frame = aFrame.frameNo;

}

fault();

}

SDL\_LockMutex(mutex1);

update = true;

SDL\_CondSignal(updateQueue);

SDL\_UnlockMutex(mutex1);

}

SDL\_WaitThread(tid, NULL);

return 0;

}

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 9 on master\***

**$** ./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: 4

New page: 3

New page: 2

New page: 1

New page: 2

New page: 3

New page: 4

New page: -99

**$** ./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

4 0 7

3 2 8

2 1 9

1 2 10

2 1 10

3 0 11

4 2 12

Terminal ending...

1. **XV6 Process Priority**
2. Giving high priority to a newly loaded process by adding a *priority* statement in *exec.c*:

…

curproc->tf->eip = elf.entry; // main

curproc->tf->esp = sp;

curproc->priority = 2; // Added statement

switchuvm(curproc);

freevm(oldpgdir);

1. Modifying *foo.c* so that the parent waits for the children:

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 += 0.001 )

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

break;

}

1. Observing the default round-robin (RR) scheduling.

$ 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 12 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 13 RUNNING 2

$ 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 14 RUNNING 2

1. Implementing Priority Scheduling in *proc.c*:

#define NULL 0

void

scheduler(void)

{

struct proc \*p;

struct proc \*p1;

struct cpu \*cpu = mycpu();

cpu->proc = 0;

for(;;){

// Enable interrupts on this processor.

sti();

struct proc \*highP = NULL;

// Loop over process table looking for process to run.

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 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){

if(p1->state != RUNNABLE)

continue;

if ( highP->priority > p1->priority ) // larger value, lower priority

highP = p1;

}

// Switch to chosen process. It is the process's job

// to release ptable.lock and then reacquire it

// before jumping back to us.

p = highP;

cpu->proc = p;

switchuvm(p);

p->state = RUNNING;

// cprintf("Process %s with pid %d running\n with createTime %d\n", p->name, p->pid, p->createTime);

swtch(&(cpu->scheduler), p->context);

switchkvm();

// Process is done running for now.

// It should have changed its p->state before coming back.

cpu-> proc = 0;

}

release(&ptable.lock);

}

}

1. Observing 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

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 12 RUNNING 2

$ nice 11 8

$ 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 RUNNABLE 10

foo 11 RUNNING 8

ps 14 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 RUNNABLE 10

foo 11 RUNNING 8

ps 15 RUNNING 2

**Discussion:** We did finished everything in this lab so we would give ourselves 20/20 points.