CS 444 Group 4

Project 1

Member:
Arvind Vepa Chauncey YAN

1 Command log for running the initial kernel

1.1 Setup the environment variables

1.1.1 bourne based shells

source /scratch/opt/environment-setup-i586-poky-linux

1.1.2 tcsh/csh shell

source /scratch/opt/environment-setup-i586-poky-linux.csh

1.2 copy the starting kernel, driver file, and other related files

cp -R /scratch/spring2015/files/ .

1.3 run the initial version of qemu

qemu-system-i386 -gdb tcp::5504 -S -nographic -kernel bzImage-qemux86.bin -drive file=core-image-lsb-sdk-qemux86.ext3,if=virtio -enable-kvm -net none -usb -localtime –no-reboot –append “root=/dev/vda rw console=ttyS0 debug”

1.4 Open gdb in a seperate terminal and boot qemu

gdb
target remote : 5504
continue

1.5 Log in with root and no password from the qemu terminal

2 Command log for running the yocto kernel within the repository

2.1 Create new repository

git init

2.2 Acquire Yocto 3.14 kernel and switch to v3.14.24 tag

git clone git://git.yoctoproject.org/linux-yocto-3.14
cd linux-yocto-3.14
git checkout tags/v3.14.24

2.3 Copy configuration file to linux-yocto

cd ..
cp config-3.14.26-yocto-qemu linux-yocto-3.14/.config

2.4 Build the new instance of kernel

cd linux-yocto-3.14
make -j4 all

2.5 Run the qemu with the new generated inmage

qemu-system-i386 -gdb tcp::5504 -S -nographic -kernel arch/x86/boot/bzImage -drive file=/scratch/spring2015/cs444-group04/core-image-lsb-sdk-qemux86.ext3,if=virtio -enable-kvm -net none -usb -localtime –no-reboot –append "root=/dev/vda rw console=ttyS0 debug"

2.6 Using gdb to boot qemu

gdb
target remote : 5504
continue

2.7 Log in with root and no password from the qemu terminal

3 Work Log

Date	Person	Task
Wed Apr 1	Arvind	pthreads
Wed Apr 1	Chauncy	random number generator
Thu Apr 2	Arvind	pthreads
Sun Apr 5	Arvind	Finished pthreads
Tue Apr 7	Arvind	Multiple threads
Tue Apr 7	Chauncy	random number generator
Wed Apr 8	Chauncy	Finished random number generator
Fri Apr 10	Arvind	Finished integrating
Sun Apr 12	Arvind	assist with group write-up
Sun Apr 12	Chauncy	set up tex file
Mon Apr 13	Chauncy	style guide
Mon Apr 13	Arvind	style guide

4 Version control log

Date	Author	Message
Fri Apr 10 21:22:11	Arvind	version 1
Fri Apr 10 23:12:42	Arvind	version 2
Fri Apr 10 23:23:42	Arvind	mersenne twister files
Sun Apr 12 20:43:22	Arvind	rdrand test
Sun Apr 12 20:52:57	Arvind	version 4 tests complete
Sun Apr 12 21:55:43	Chauncey	late adding latex example
Sun Apr 12 21:57:24	Chauncey	Working on the groupwriteup
Sun Apr 12 21:58:22	Chauncey	Finished detecting rdrand support and using rdrand and mt

5 Concurrency

In this solution, we create two named semaphores and then a number of pthreads, corresponding to the threads specified on the command line (the first number is the number of producer threads an the second number is the number of consumer threads). After these threads are created, the threads are joined, so when the threads are terminated, they are joined.

In the file, we have a static member that corresponds to the pthread_mutex lock and we have the initial declaration of the consumer and producer method, along with the declaration for the struct that contains both the number that will be printed out as well as the amount of time that it will wait.

In order to check if the system support rdrand instruction of not, check_rd() is called. It runs asm code "cpuid %0, which return the cpu architecture information to a variable in terms of binary. Specifically, CPUID was called with register EAX = 1. ECX will return a 32 bits data that include all cpu info. If bit 30 is return 1 then the system support rdrand.

When the system has rdrand support, rnd_int() is going to generate an unsigned int using the rdrand method. The way to using rdrand is through asm code. it generate an unsigned long integer from cpu and store it in the variable.

When the system has no rdrand support, such as OS-class, rnd_int() is going to generate an unsigned int using Mersenne Twister method. First, we need to initialize mt with a seed array. Then we call genrand_int32() to get a random unsigned integer.

Within the consumer_problem method, there is a do-while loop that runs infinitely. As specified in the directions, if the buffer is empty, then sem_wait is called on the semaphore and the thread waits. Once the producer_problem method calls sem_post from the other method after adding an item to the buffer will the semaphore be released.

After the semaphore is released, the pthread is locked, and then the thread sleeps for the time specified within the first element. The number that is assigned to the first element is then printed. After this item is consumed, the element is overwritten by all the previous elements within the buffer, and then the semaphore that is used by the producer problem in order to signify that the buffer is ready to add an additional item when filled is posted, so that the producer may add another item. After the first item is removed from the buffer, the pthread is unlocked, and the do-while loop runs for a second iteration, and all subsequent iterations.

For the producer_problem method, there is a do-while loop that runs infinitely. As specified in the directions, if the buffer is full, then sem_wait is called on the semaphore and the thread waits. Once the consumer_problem method calls sem_post from the other method after removing an item from the buffer will the semaphore be released.

After the semaphore is released, the random number generator is initialized and the producer thread sleeps for a time between 2-7 seconds. Then the pthread is locked. Afterwards, then the next empty element of the buffer is filled with the struct containing the two random numbers and then the semaphore used by the consumer problem in order to signify that the buffer is ready to remove an additional item when the buffer is empty is posted, so that the consumer may remove another item. After the item is added to the buffer, the pthread is unlocked, and the do-while loop runs for a second iteration, and all subsequent iterations.

Appendix 1: Source Code

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <pthread.h>
#include "mt19937ar.h"
#include <fcntl.h>
#include <semaphore.h>
#include <sys/stat.h>

static pthread_mutex_t mutex_sum;

extern void *consumer_problem(void *a);
extern void *producer_problem(void *a);
```

```
sem_t *s1;
sem_t *s2;

struct t_elements {
        int num;
        int sec;

};

struct t_elements buffer[32];

unsigned long rnd_int()
{
        if (check_rd() == 1 ){ //rdrand()
                unsigned long ul = 0ul;
                __asm__ __volatile__(
                                "rdrand %0"
                                :"=r"(ul)
                                );
                return ul;
        } else {
                return genrand_int32();

        }
}
int check_rd()
{
        unsigned long eax = 1, ecx,ebx;
        __asm__ __volatile__(
                        "cpuid"
                        :"=a"(eax),
                        "=b"(ebx),
                        "=c"(ecx)
                        :"0"(eax),
                        "2"(ecx)
                        );
        if ((ecx>>30)&0x01 == 1)
                return 1;

        return 0;
}

void *consumer_problem(void *a)
{
        do {
                int i = 0;
                int z = 0;
                if (buffer[0].num == NULL) {
                        sem_wait(s2);
                }
```

```
		pthread_mutex_lock(&mutex_sum);
		sleep(buffer[0].sec);
		printf("The number is: %d\n", buffer[0].num);
		fflush(stdout);

		for (z = 31; z >= 0; z--) {
			if (buffer[z].num != NULL) {
				memcpy(buffer, &buffer[1],
						z*sizeof(struct t_elements));
				buffer[z].num = NULL;
				buffer[z].sec = NULL;
				sem_post(s1);
				break;
			}
		}

		pthread_mutex_unlock(&mutex_sum);
	}
	while (1);
}
void *producer_problem(void *a)
{
	do {
		if (buffer[31].num != NULL) {
			sem_wait(s1);
		}
		int seed;
		init_genrand((unsigned long)&seed);
		int j = 0;
		sleep(rnd_int()%5+3);
		pthread_mutex_lock(&mutex_sum);

		for (j = 0; j < 32; j++) {
			if (buffer[j].num == NULL) {
				unsigned long init[4] = {0x123, 0x234,
							0x345, 0x456}, length = 4;

				buffer[j].num = rnd_int() % 8 + 2;
				fflush(stdout);
				buffer[j].sec = rnd_int() % 8 + 2;
				fflush(stdout);
				sem_post(s2);
				break;
			}
		}
		pthread_mutex_unlock(&mutex_sum);
	}
	while (1);
}

int main(int argc, char *argv[], char *envp[])
```

```
{
        char *c = "semaphore_1";
        char *b = "semaphore_2";

        s1 = sem_open(c, O_CREAT, 0666, 0);
        s2 = sem_open(b, O_CREAT, 0666, 0);
        int num1 = atoi(argv[1]);
        int num2 = atoi(argv[2]);
        int s = 0;
        int z = 0;
        pthread_t *threads;

        threads = (pthread_t *) malloc((num1 + num2) * sizeof(pthread_t));
        pthread_mutex_init(&mutex_sum, NULL);
        for (s = 0, z = 0; (s + z) < (num1 + num2);) {
                if (s < num1) {
                        pthread_create(&threads[s], NULL,
                                       producer_problem,NULL);
                        s++;
                }
                if (z < num2 ) {
                        pthread_create(&threads[num1 + z], NULL,
                                       consumer_problem,NULL);
                        z++;
                }
        }
        for (s = 0; s < (num1 + num2); s++) {
                pthread_join(threads[s], NULL);
        }
}
```