Collin Gros
11-05-2020
cs474
Project 2

PROJECT 2 REPORT

DESCRIPTION

The objective of this project is to use semaphores to protect a limited size resource. A circular buffer is served as the limited size shared resource with 15 positions, with each position storing 1 character. Two threads, a producer and consumer, are used to store and retrieve data from this circular buffer. The producer reads a file, and stores data in the buffer. The consumer pops data from the buffer, and writes it to the screen. The consumer sleeps for 1 second in between reads.

CODE

circ.c

```
/*      collin gros
        11-04-2020
        cs474
        proj2

        this code was retrieved from:
                https://stackoverflow.com/questions/827691/how-do-you-
                        implement-a-circular-buffer-in-c

        and is used by main.c for the circular buffer.     */

#include <string.h>
#include <stdlib.h>

#include "circ.h"


void cb_init(circular_buffer *cb, size_t capacity, size_t sz)
{
    cb->buffer = malloc(capacity * sz);
    if(cb->buffer == NULL)
        // handle error
    cb->buffer_end = (char *)cb->buffer + capacity * sz;
    cb->capacity = capacity;
    cb->count = 0;
    cb->sz = sz;
    cb->head = cb->buffer;
    cb->tail = cb->buffer;
}

void cb_free(circular_buffer *cb)
{
    free(cb->buffer);
    // clear out other fields too, just to be safe
}

void cb_push_back(circular_buffer *cb, const void *item)
{
    if(cb->count == cb->capacity){
        // handle error
```

```
    }
    memcpy(cb->head, item, cb->sz);
    cb->head = (char*)cb->head + cb->sz;
    if(cb->head == cb->buffer_end)
        cb->head = cb->buffer;
    cb->count++;
}

void cb_pop_front(circular_buffer *cb, void *item)
{
    if(cb->count == 0){
        // handle error
    }
    memcpy(item, cb->tail, cb->sz);
    cb->tail = (char*)cb->tail + cb->sz;
    if(cb->tail == cb->buffer_end)
        cb->tail = cb->buffer;
    cb->count--;
}
```

circ.h

```
/*      collin gros
        11-04-2020
        cs474
        proj2

        this code was retrieved from:
                https://stackoverflow.com/questions/827691/how-do-you-
                        implement-a-circular-buffer-in-c

        and is used by main.c for the circular buffer.     */

#ifndef __CIRCULAR_BUFFER_INCLUDED__
#define __CIRCULAR_BUFFER_INCLUDED__

#include <string.h>
#include <stdlib.h>


typedef struct circular_buffer
{
    void *buffer;     // data buffer
    void *buffer_end; // end of data buffer
    size_t capacity;  // maximum number of items in the buffer
    size_t count;     // number of items in the buffer
    size_t sz;        // size of each item in the buffer
    void *head;       // pointer to head
    void *tail;       // pointer to tail
} circular_buffer;

void cb_init(circular_buffer *cb, size_t capacity, size_t sz);

void cb_free(circular_buffer *cb);

void cb_push_back(circular_buffer *cb, const void *item);

void cb_pop_front(circular_buffer *cb, void *item);


#endif
```

main.c

```
/*
	collin gros
	11-03-2020
	cs474
	project2

	purpose:	learn how to use semaphores to protect a limited size
			resource.

	references:
			given: textbook
			given: CS474_project2.pdf
			https://www.geeksforgeeks.org/use-posix-semaphores-c/
			https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem
			https://stackoverflow.com/a/827749

*/

#define _REENTRANT
/*	size of our shared circular buffer  */
#define MAX_BUFFER_SIZE	15
/*	character size is 1 byte	*/
#define BUFFER_SZ		1

#include <pthread.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <semaphore.h>
#include <unistd.h>

/*	circular_buffer data structure	*/
#include "circ.h"


/*	semaphore for ensuring mutual exclusion in consumer and producer */
sem_t mutex;
/*	number of items in buffer	*/
sem_t fill_count;
/*	number of space in buffer	*/
sem_t empty_count;

/*	allocate circular buffer in data segment (global) so it is not
	in the stack since the stack is not shared across threads */
static circular_buffer buf;


/*	producer thread:
		reads characters one by one from mytest.dat and places it in
		the buffer and continues to do that until EOF. it informs
		the consumer when reaching EOF (after placing last char in buff).
		it does this by placing * into the buffer.  */
void* producer(void* arg)
{
	/*	open test file for reading   */
	char c;
	FILE *fp;
```

```
        fp = fopen("mytest.dat", "r");
        while (fscanf(fp, "%c", &c) != EOF) {
                /*      wait until there is space in buffer  */
                sem_wait(&empty_count);
                /*      wait our turn  */
                sem_wait(&mutex);

                /*      inserting character into shared buffer      */
                cb_push_back(&buf, &c);

                /*      end our turn   */
                sem_post(&mutex);
                /*      for consumer to know it has available data to get  */
                sem_post(&fill_count);
        }

        fclose(fp);

        /*      wait until there is space in buffer  */
        sem_wait(&empty_count);
        /*      wait our turn  */
        sem_wait(&mutex);

        /*      insert * into buffer as we've reached EOF   */
        const char star = '*';
        cb_push_back(&buf, &star);

        /*      end our turn   */
        sem_post(&mutex);
        /*      for consumer to know it has available data to get  */
        sem_post(&fill_count);
}


/*      consumer thread:
                runs slower than the producer (1 second sleep in between reads of
                buffer).
                reads chars one by one from the shared buffer and prints to
                the screen.    */
void* consumer(void* arg)
{
        while (1) {
                /*      sleep instruction added to run slower than producer */
                sleep(1);

                /*      wait until producer gives us data    */
                sem_wait(&fill_count);
                /*      wait for our turn     */
                sem_wait(&mutex);

                /*      retrieve char from buffer and print it      */
                char c;
                cb_pop_front(&buf, &c);

                /*      abort if EOF   */
                if (c == '*') {
                        break;
                }

                printf("%c", c);
                fflush(stdout);
```

```
                /*      end our turn   */
                sem_post(&mutex);
                /*      let producer know there is an available spot       */
                sem_post(&empty_count);
        }
}


/*      parent process (main):
                creates both the producer and consumer threads and waits until
                both are finished tod estroy semaphores.    */
int main()
{
        /*      initialize the circular buffer with a maximum size and the
                size of the data type */
        cb_init(&buf, MAX_BUFFER_SIZE, BUFFER_SZ);

        /*      initialize semaphore to 1 so the first created process (producer)
                executes first */
        sem_init(&mutex, 0, 1);
        /*      no spots are full     */
        sem_init(&fill_count, 0, 0);
        /*      all spots are empty   */
        sem_init(&empty_count, 0, MAX_BUFFER_SIZE);


        /*      create producer and consumer threads. producer runs critical
                section first. */
        pthread_t tprod, tcons;
        pthread_create(&tprod, NULL, producer, NULL);
        pthread_create(&tcons, NULL, consumer, NULL);

        /*      wait until both are finshed  and destroy producer and consumer
                threads.       */
        pthread_join(tprod, NULL);
        pthread_join(tcons, NULL);

        /*      cleanup*/
        sem_destroy(&mutex);
        sem_destroy(&fill_count);
        sem_destroy(&empty_count);

        return 0;
}
```

makefile

```
# collin gros
# 11-03-2020
# cs474
# project2
#
# makefile for project 2
# compile with:
#               make
# clean with:
#               clean


all: circ.o
        gcc main.c circ.o -lpthread -lrt -o run
circ.o:
        gcc -c circ.c
```

```
clean:
	rm -f run
	rm -f *.o
```

mytest.dat

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

OUTPUT

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

[note: one 'x' is output every second]
```

ANALYSIS

The program takes about 2 minutes, 34 seconds to complete. Every second, one character from *mytest.dat* is written to the screen. In *main.c*, this is achieved by having the producer insert characters one-by-one into the circular buffer (which is a shared data structure in the Data segment of memory), while the consumer retrieves them. The consumer is restricted in the sense that it must wait 1 second between each read. This lets us observe how the output is being printed to the screen, rather than everything being output at once.

Three semaphores are used; *fill_count*, *empty_count*, and *mutex*. *fill_count* is the number of items inside of the buffer. This is used by the producer to signal the consumer to start. *empty_count* is the number of empty space inside of the buffer. This is used by the consumer to signal the producer to start. *mutex* is used for ensuring the circular buffer is not used at the same time by the two threads. According to Wikipedia, *mutex* should generally not be used with a semaphore, but I chose to anyway because it seemed to work correctly (Wikipedia contributors, 2020). I used Wikipedia's solution to the producer-consumer problem as the algorithm for *main.c*.

circ.c and *circ.h* were both developed from a stackoverflow question and implemented in my assignment, as it was easier to find someone's implementation of a circular buffer, rather than creating my own (Rosenfield, 2009). Permission was obtained by the professor to use this code.

WORKS CITED

Wikipedia contributors. (2020, May 27). Producer–consumer problem. In Wikipedia, The Free Encyclopedia. Retrieved 18:11, November 5, 2020, from https://en.wikipedia.org/w/index.php?title=Producer%E2%80%93consumer_problem&oldid=959132413

Rosenfield, Adam. (2009, May 6). How do you implement a circular buffer in C?. stackoverflow. Retrieved 18:11, November 5, 2020, from https://stackoverflow.com/a/827749