ELEC 377 Lab 3: Description of the Program

Name & Student ID: Shiyan Boxer (20106887) and Arsh Kochhar (20104779)
Date: Nov 2nd, 2020

Overview of the Program

This program implements hardware busy wait for the critical section problem, allowing communication between user level processes using shared memory. The lab was programmed using virtual memory capabilities of Linux to map a single frame from physical memory to virtual address space of multiple processes.

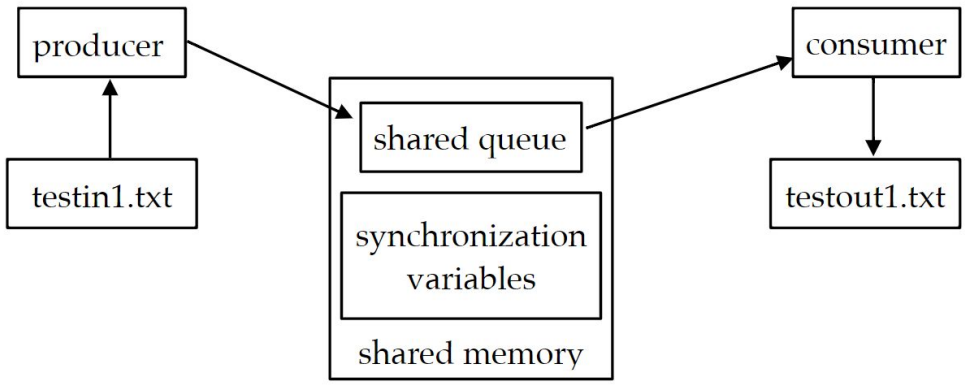
Program Files

1. **Makefile** - Builds the system.
2. **meminit.c** Compile to meminit
3. **producer.c** - Code for the producer
4. **consumer.c** - Code for the consumer
5. **common.h** - Contains the definitions of the struct declaration used to impose structure on the shared memory.
6. **common.c** - Contains the code for the getMutexand releaseMutexroutines

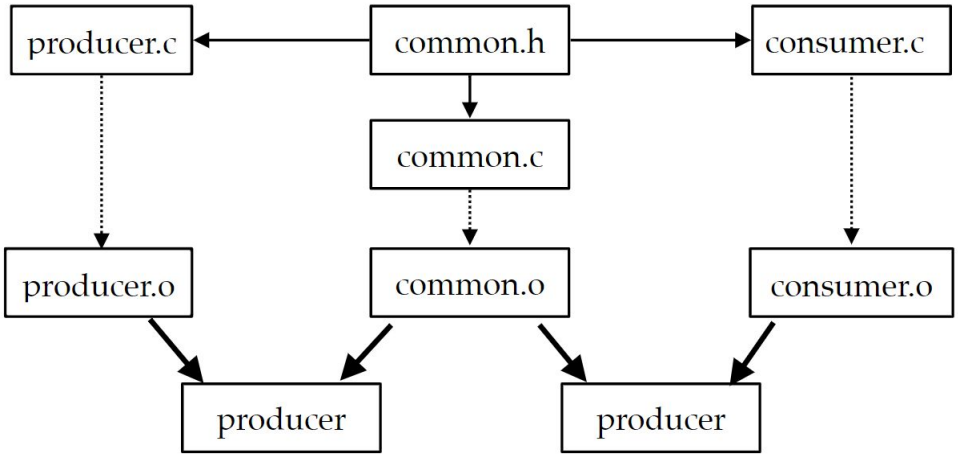
Synchronization

The program implements the hardware algorithm shown on OnQ to allow synchronization between two processes. A producer program copies standard input one byte at a time into a shared buffer and a consumer program retrieves a single character at a time from the buffer and writes it to the standard output.

The general structure of the program is represented in the following diagram.



The build process is shown in the following diagram.



Solution of our Program

The solution of the program will be explained by describing each file and its functionality.

makefile

The **makefile** is used to build the system. When using the command “make”, it builds the system.

```
all: consumer meminit producer
common.o: common.h
consumer: consumer.c common.o common.h
	cc -o consumer consumer.c common.o
meminit: meminit.c
	cc -o meminit meminit.c
producer: producer.c common.o common.h
	cc -o producer producer.c common.o
```

meminit.c

The **meminit.c** file was provided and was not modified. Once compiled, running the program creates the shared memory segment (200 bytes) and initializes it to NULL values. This was run between each test to reinitialize the shared memory segment.

producer.c

The **producer.c** file contains code for the producer program. It includes the code to access the shared memory segment and map it into memory. It also copies the input to the shared memory one byte at a time.

First, the producer increments the number of producers that are accessing the shared buffer. The number of producers is a shared resource, the increment must be guarded by calls to getMutex and releaseMutex.The producer contains a loop that reads each character (using getchar()) until it reaches the end of file (getchar() returns EOF). Since the solution uses busy waiting and does not put a process to sleep, we used an inner loop that requests access to the critical section (using getMutex) then checks to see if there is room in the queue. If there is, it adds the character to the queue. It then releases the mutex. The inner loop is controlled by a flag that is set whenever the producer succeeds in adding a byte to the buffer.
The end of the file (EOF) condition is identified by adding a flag to the buffer that the producer can set to indicate that the end of the data has been reached. The algorithm for the producer used in this program is the following:

```
getMutex(&sharedPtr->lock);
      sharedPtr->numProducers++;
      releaseMutex(&sharedPtr->lock);

      while ((c = getchar())!=EOF){ //getChar allows to check
till the end of file
            stored=FALSE;
            while(stored==FALSE){ //checking to see if there is
room in the queue
                  getMutex(&sharedPtr->lock);
                        if(sharedPtr->count<BUFFSIZE){
                              sharedPtr->buffer[sharedPtr->in]= c;
//add character to the queue
                              sharedPtr->in = (sharedPtr->in +
1)%BUFFSIZE;
                              sharedPtr->count++;
                              stored=TRUE;
                        }
                        releaseMutex(&sharedPtr->lock); //releases
the mutex
                  }
            }

            getMutex(&sharedPtr->lock);
            sharedPtr->numProducers--;
            releaseMutex(&sharedPtr->lock);

            return 0;
```

consumer.c

The **consumer.c** file contains code for the consumer program. It has the same code to map the shared memory segment into memory as the producer.c file. It also copies the data from the shared memory to the output one byte at a time.

It uses nested loops, with a flag to indicate when it has successfully retrieved a byte from the buffer. It exists when the number of producers is zero. The algorithm for the consumer used in this program is the following:

```
getMutex(&sharedPtr->lock);
      int numProd=sharedPtr->numProducers;
      releaseMutex(&sharedPtr->lock);

      int charRead= TRUE;

      while(numProd>0 && charRead){ //while charRead and
producer exists
            charRead=FALSE;
            while (charRead == FALSE && numProd > 0){ //while
charRead is false and there is a producer
                  getMutex(&sharedPtr->lock);
                  if(sharedPtr->count>0){ //if there is a char in
the buffer
                        c=sharedPtr->buffer[sharedPtr->out];
//retrieve and store in C

sharedPtr->out=(sharedPtr->out+1)%BUFFSIZE;
                        charRead=TRUE; //set to true
                        sharedPtr->count--; //decremenet couunter
                  }i
                  else{
                        numProd=sharedPtr->numProducers; // if the
buffer is empty, update producers (local variable)
                  }
                  releaseMutex(&sharedPtr->lock); // release
mutex
            }
            putchar(c); //stdout
      }
      return 0;
```

common.h

The **common.h** file contains the definitions of the struct declaration, shared memory segment and prototypes for semaphore functions used to impose structure on the shared memory. The size of the data structure is less than, or equal to, the size of the shared segment (200 bytes). It is included (using the #include directive) into both producer.c and consumer.c and into common.c. Also, buffer size was kept small (5 bytes) to allow for proper testing.

```
#define MEMSIZE 200
#define BUFFSIZE 5

struct shared {
    char buffer[BUFFSIZE];
    int numProducers;
    int lock;
    int out;
    int in;
    int count;
};

void getMutex(int * lock);
void releaseMutex(int * lock);
```

common.c

The **common.c** file contains code for the getMutex and releaseMutex routines. The entry and exit code for the critical section in these two routines were implemented. The consumer process reads data (chars in this case) from the shared buffer and puts them in the stdout in simple terms, consuming the data from the buffer.

```
int test_and_set(int * lock){
    return __cmpxchg(lock,0,1,4);
}

void getMutex(int *  lock){
     // this should not return until it has mutual exclusion.
Note that many versions of
     // this will probobly be running at the same time.
     while (test_and_set(lock));
     return;
}

void releaseMutex(int * lock){
     // set the mutex back to initial state so that somebody
else can claim it
     *lock = FALSE;
```

```
        return;
}
```