GEBZE TECHNICAL UNIVERSITY

CSE 344- SYSTEMS PROGRAMMING

HW-4 REPORT

ESRA NUR ARICAN

161044028

OBJECTIVE

There is one supplier thread and multiple consumer threads. The supplier brings materials, one by one. And the consumers consume them, two by two. Each actor will be modeled by its own thread.

./hw4 -C 10 -N 5 -F inputfilePath

SOLUTION

Functions that I used in this homework:

```
void* supplierFunc(void * inputFilePath);
void* consumerFunc(void * ind);
void print_string(char string[]);
void print_error(char error_message[]);
void signalHandler(int sig);
```

Main function:

In the main function, first I assigned commandline parameters to variables C,N, and filePath. The number of arguments entered here, the value of C greater than 4 and the value of N greater than 1, and the use of the program were checked.

In the beginning of the main function, ı used setbuf(stdout,NULL); to print without buffering.

Later, I defined the semaphore as system V semaphores will be used in this assignment.

A semaphore set containing 2 semaphores was created with System V semaphores. 2 semaphores are used for value ‘1’ and value ‘2’.

```
//System V semaphore
union semun
{
    int val;
    struct semid_ds *buf;
    ushort array [1];
} sem_attr;

semid = semget( IPC_PRIVATE,2, 0666 | IPC_CREAT);
if (semid == -1){
    perror("semid semget() error");
    exit(EXIT_FAILURE);
}

sem_attr.val = 0;
if (semctl( semid, 0,SETVAL, sem_attr) == -1){
    perror("error on semctl()");
    exit(EXIT_FAILURE);
}
if (semctl( semid, 1, SETVAL, sem_attr) == -1){
    perror("error on semctl()");
    exit(EXIT_FAILURE);
}
```

After creating the necessary variables for the signal handler in the main function, I moved on to creating the threads that will do the main work.

I created 1 detached supplier thread and consumer threads as many as C. Since detached threads are not joinable, I only joined consumer threads here.

```
    //creating supplier thread.SUPPLIER IS A DETACHED THREAD
    pthread_t supplier_thread;
    pthread_attr_t detachedThread;
    pthread_attr_init(&detachedThread);
    pthread_attr_setdetachstate(&detachedThread, PTHREAD_CREATE_DETACHED);

    if(pthread_create(&supplier_thread, &detachedThread, supplierFunc, inputFilePath) != 0) {
      perror("ERROR pthread_create ");
      exit(EXIT_FAILURE);
    }

    pthread_attr_destroy(&detachedThread);

    //creating consumer threads
    int *index;
    int i=0;
    pthread_t consumer_threads[C];

    for(i = 0; i < C; i++) {
        index=&i;
        if(pthread_create(&consumer_threads[i], NULL, consumerFunc, index) != 0) {
          perror("ERROR pthread_create consumers");
          exit(EXIT_FAILURE);
        }
    }

    threadSig=consumer_threads;
```

The main function terminates its process after finally clearing the semaphores.

```
    for(i = 0; i < C; i++) {
        if(pthread_join(consumer_threads[i],NULL) != 0) {
          perror("ERROR pthread_create consumers");
          exit(EXIT_FAILURE);
        }
    }

    //remove semaphores
    if (semctl (semid, 0, IPC_RMID) == -1) {
        perror ("semctl IPC_RMID");
        exit (1);
    }

    pthread_exit(0);
}
```

Supplier Function:

The supplier function should read the materials from the file one by one, change the values that represent 1 and 2 in the semaphore, and print some print statements.

For this, I first determined the time variables to be used in each print. I then set a variable of type struct sembuf to use in the semaphore operation and initialize its values. I initially set the value of sem_op to 1 as its value should increase by 1 each time the supplier reads a new material.

Then, I read a character with a loop until I reach the end of the file, and if the character read with two if conditions is 1, I printed the semaphore operation and prints for it, and if 2, I printed the semaphore operations and prints for it.

In this if condition, first the message that the character has been read is printed , then the semaphore value is increased, and finally the message that the material has been delivered is printed

```
    while(1){
        while(((bytesread = read(fdInput, buf, 1)) == -1) &&(errno == EINTR));
        if(bytesread <= 0)
            break;
        if(buf[0]=='1'){
            asem.sem_num= 0;  //semaphore that represents 1s
            printf("[%19.s] Supplier: read from input a '1'. Current amounts %d x '1', %d x '2'.\n"
                    ,ctime(&t),semctl(semid,0,GETVAL),semctl(semid,1,GETVAL));

            /* post operation */
            asem .sem_op = 1;
            if (semop (semid, &asem, 1) == -1) {
                print_error ("supplierFunc | semop error\n");
                exit (EXIT_FAILURE);
            }

            printf("[%19.s] Supplier: delivered a '1'. Current amounts %d x '1', %d x '2'.\n"
                    ,ctime(&t),semctl(semid,0,GETVAL),semctl(semid,1,GETVAL));
        }
        if(buf[0]=='2'){
            asem.sem_num=1; //semaphore that represents 2s
            printf("[%19.s] Supplier: read from input a '2'. Current amounts %d x '1', %d x '2'.\n"
                    ,ctime(&t),semctl(semid,0,GETVAL),semctl(semid,1,GETVAL));

            /* post operation */
            asem .sem_op = 1;
            if (semop (semid, &asem, 1) == -1) {
```

Consumer Function:

The consumer function consumes the materials 2 by 2, unlike the supplier. So both character 1 and 2 semaphores must contain them. Therefore, in order to provide this control, I kept an array in the form of struct sembuf asem[2] in the consumer function so that I could control both materials.

asem[0] represents ‘1’ and asem[1] represents ‘2’s.

Since the task of the consumer function is to consume the materials by looping through C, I first checked whether both the materials were present in the for loop. Then I wrote the print statements on the screen and performed a wait operation to represent that the materials were consumed.

```
for(int i=0;i<N;i++){

    //before consuming
    sprintf(message,"[%.19s] Consumer-%d at iteration %d (waiting). Current amounts: %d x '1', %d x '2'\n",
        ctime(&t),index,i,semctl(semid,0,GETVAL),semctl(semid,1,GETVAL));

    print_string(message);

    //wait operation
    if (semop (semid, asem, 2) == -1) {
        print_error ("consumerFunc| semop error\n");
        exit (EXIT_FAILURE);
    }

    //after consuming
    sprintf(message,"[%.19s] Consumer-%d at iteration %d (consumed). Post-consumption amounts: %d x '1', %d x '2'\n"
        ,ctime(&t),index,i,semctl(semid,0,GETVAL),semctl(semid,1,GETVAL));

    print_string(message);

}
sprintf(message,"[%.19s]Consumer-%d has left.\n",ctime(&t),index);
print_string(message);
```

The print_string function was written to be able to print a message to the screen.

The print_error function was written to print error messages on the screen.

The SignalHandler function was written to handle the CTRL-c signal.

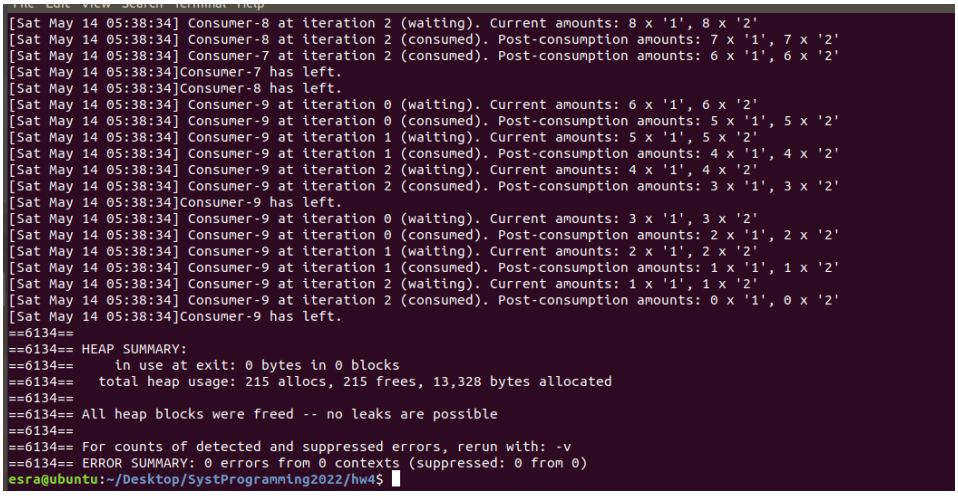
```c
// Prints string in the STDOUT
void print_string(char string[]){
    if(write(STDOUT_FILENO,string,strlen(string))<0){
        perror("print_string | write error\n");
        exit(EXIT_FAILURE);
    }
}

//Prints error in the STDERR
void print_error(char error_message[]){
    if(write(STDERR_FILENO,error_message,strlen(error_message))<0){
        exit(EXIT_FAILURE);
    }
}

// signal handler for SIGINT
void signalHandler(int sig){
    if (sig==SIGINT && sigintFlag!=1){
        sigintFlag=1;
        if (threadSig!=NULL){
            for (int i = 0; i < C; i++)
                pthread_kill(threadSig[i],SIGINT);
        }
    }
    printf("\nCase CTRL-C: The program was terminated.\n");
}
```

Makefile with wall and clean :

```
makefile   hw4.c   untitled
1  CC = gcc
2  CFLAGS = -Wall -pedantic-errors -std=gnu99
3  LIBS = -pthread -lrt
4  OBJ = hw4.o
5
6  %.o: %.c
7      $(CC) -c -o $@ $< $(CFLAGS) $(LIBS)
8
9  all: hw4
10
11 hw4: $(OBJ)
12     $(CC) -o $@ $^ $(CFLAGS) $(LIBS)
13
14
15 clean:
16     rm -f *.o
```

Program output with no warnings and no errors, memory leaks with valgrind:

```
esra@ubuntu:~/Desktop/SystProgramming2022/hw4$ make clean
rm -f *.o
esra@ubuntu:~/Desktop/SystProgramming2022/hw4$ make
gcc -c -o hw4.o hw4.c -Wall -pedantic-errors -std=gnu99 -pthread -lrt
gcc -o hw4 hw4.o -Wall -pedantic-errors -std=gnu99 -pthread -lrt
esra@ubuntu:~/Desktop/SystProgramming2022/hw4$ valgrind ./hw4 -C 10 -N 3 -F in10x3.txt
==6134== Memcheck, a memory error detector
==6134== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==6134== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==6134== Command: ./hw4 -C 10 -N 3 -F in10x3.txt
==6134==
[Sat May 14 05:38:34] Consumer-2 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'
[                   ] Supplier: read from input a '1'. Current amounts 0 x '1', 0 x '2'.
[Sat May 14 05:38:33] Consumer-0 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'
[Sat May 14 05:38:34] Consumer-2 at iteration 0 (waiting). Current amounts: 0 x '1', 0 x '2'
[                   ] Supplier: delivered a '1'. Current amounts 1 x '1', 0 x '2'.
[                   ] Supplier: read from input a '1'. Current amounts 1 x '1', 0 x '2'.
[                   ] Supplier: delivered a '1'. Current amounts 2 x '1', 0 x '2'.
[                   ] Supplier: read from input a '2'. Current amounts 2 x '1', 0 x '2'.
[Sat May 14 05:38:34] Consumer-2 at iteration 0 (consumed). Post-consumption amounts: 1 x '1', 0 x '2'
[Sat May 14 05:38:34] Consumer-2 at iteration 1 (waiting). Current amounts: 1 x '1', 0 x '2'
[                   ]Supplier: delivered a '2'. Current amounts 1 x '1', 0 x '2'.
[                   ] Supplier: read from input a '1'. Current amounts 1 x '1', 0 x '2'.
[                   ] Supplier: delivered a '1'. Current amounts 2 x '1', 0 x '2'.
[                   ] Supplier: read from input a '1'. Current amounts 2 x '1', 0 x '2'.
[                   ] Supplier: delivered a '1'. Current amounts 3 x '1', 0 x '2'.
[Sat May 14 05:38:34] Consumer-4 at iteration 0 (waiting). Current amounts: 3 x '1', 0 x '2'
[                   ] Supplier: read from input a '2'. Current amounts 3 x '1', 0 x '2'.
[                   ]Supplier: delivered a '2'. Current amounts 2 x '1', 0 x '2'.
```


[Sat May 14 05:38:34] Consumer-8 at iteration 2 (waiting). Current amounts: 8 x '1', 8 x '2'
[Sat May 14 05:38:34] Consumer-8 at iteration 2 (consumed). Post-consumption amounts: 7 x '1', 7 x '2'
[Sat May 14 05:38:34] Consumer-7 at iteration 2 (consumed). Post-consumption amounts: 6 x '1', 6 x '2'
[Sat May 14 05:38:34]Consumer-7 has left.
[Sat May 14 05:38:34]Consumer-8 has left.
[Sat May 14 05:38:34] Consumer-9 at iteration 0 (waiting). Current amounts: 6 x '1', 6 x '2'
[Sat May 14 05:38:34] Consumer-9 at iteration 0 (consumed). Post-consumption amounts: 5 x '1', 5 x '2'
[Sat May 14 05:38:34] Consumer-9 at iteration 1 (waiting). Current amounts: 5 x '1', 5 x '2'
[Sat May 14 05:38:34] Consumer-9 at iteration 1 (consumed). Post-consumption amounts: 4 x '1', 4 x '2'
[Sat May 14 05:38:34] Consumer-9 at iteration 2 (waiting). Current amounts: 4 x '1', 4 x '2'
[Sat May 14 05:38:34] Consumer-9 at iteration 2 (consumed). Post-consumption amounts: 3 x '1', 3 x '2'
[Sat May 14 05:38:34]Consumer-9 has left.
[Sat May 14 05:38:34] Consumer-9 at iteration 0 (waiting). Current amounts: 3 x '1', 3 x '2'
[Sat May 14 05:38:34] Consumer-9 at iteration 0 (consumed). Post-consumption amounts: 2 x '1', 2 x '2'
[Sat May 14 05:38:34] Consumer-9 at iteration 1 (waiting). Current amounts: 2 x '1', 2 x '2'
[Sat May 14 05:38:34] Consumer-9 at iteration 1 (consumed). Post-consumption amounts: 1 x '1', 1 x '2'
[Sat May 14 05:38:34] Consumer-9 at iteration 2 (waiting). Current amounts: 1 x '1', 1 x '2'
[Sat May 14 05:38:34] Consumer-9 at iteration 2 (consumed). Post-consumption amounts: 0 x '1', 0 x '2'
[Sat May 14 05:38:34]Consumer-9 has left.
==6134==
==6134== HEAP SUMMARY:
==6134==   in use at exit: 0 bytes in 0 blocks
==6134==   total heap usage: 215 allocs, 215 frees, 13,328 bytes allocated
==6134==
==6134== All heap blocks were freed -- no leaks are possible
==6134==
==6134== For counts of detected and suppressed errors, rerun with: -v
==6134== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
esra@ubuntu:~/Desktop/SystProgramming2022/hw4$