Göktuğ Ali Akın
161044018

CSE 344
SYSTEM PROGRAMMING

PROJECT 4 REPORT

Problem definition

This project includes cigarette-smokers derive synchronization problem. There are two actors here, one is wholesaler which delivers ingredients to chefs, and there are instances of chefs that get delivered items from wholesaler to prepare desert, just like cigarette-smokers problem. Chefs needs 4 items to prepare desert, every chefs has two distinct items in infinite amount.

There are 6 chefs that waits for 2 ingredients to prepare desert. Chefs were represented as 6 different threads, and wholesaler was represented as one thread which is actually main thread.

Starting threads

Chefs were represented as 6 different threads, wholesaler was represented as one thread which is actually main thread.

In order to store the chef threads to access them later, global pthread_t array used ; Code snippet of startin threads ;

```
for (int i = 0; i < NUM_OF_CHEFS; ++i)
{
        chef_nums[i] = i+1;
        pthread_create(&(chefs[i]),NULL,run_chef,&(chef_nums[i]));
}
```

Chef nums is global array, represents the chef's number. It is globally declared since If it was locally declared, values of chef numbers deleted after scope finishes.

Synchronization with semaphores

Just like cigarette-smokers problem, we need System V semaphores here in order to get two items at the **same** time. Using the System V semaphores in this kind of scenarios, it is easy to implement, posix variant of this problem implementation is more challenging than this.

Initializing the System V semaphores

In order the initialize and get the System V semaphores, I used **semget()** system call.

- **The semget() system call returns the System V semaphore set identifier associated with the argument key.**[1]

By this **semget()** system call, I got a semaphore set which contains 4 semaphores that represents the ingredients (flour,walnut,sugar and milk).

all_sems = semget(IPC_PRIVATE,5,0660 | IPC_CREAT); // all_sem is global variable.

Last semaphore, whose number is 4, represent the desert_posted or not. By this semaphore, the wholesaler wait until desert prepared by any chef. The first four semaphores (0,1,2,3) represents the ingredients.

In order to give initial values to the semaphores, I used **semctl()** system call.

- **semctl() performs the control operation specified by cmd on the System V semaphore set identified by semid.**[2]

Code snippet of initializing semaphores ;

```
union semun arg; /* Fourth argument for semctl() */
unsigned short init_sem_vals[5];
for (int i = 0; i < 5; ++i)
        init_sem_vals[i] = 0;
arg.array = init_sem_vals;
```

semun union type must be declared by the programmer. This type is being used in semctl() function initialize semaphore.

[1] Linux manual page
[2] Linux manual page

Semaphore operations

Chefs, must get two items that he/she needs at the same time. This operation can be done by System V semaphores through **semop()** system call.

- **semop() performs operations on selected semaphores in the set indicated by semid through argument whose type is sembuf struct array.**[3]

e.g : waiting wulnat and sugar in the chef 1 :

```
// wait (get) W
sops[0].sem_num = 2;
sops[0].sem_op = -1;
sops[0].sem_flg = 0;
// wait (get) S
sops[1].sem_num = 3;
sops[1].sem_op = -1;
sops[1].sem_flg = 0;
```

after initializing **struct sembuf** array that represents the operations, operation can be done by **semop()** system call ;

```
semop(all_sems,sops,2) // all_sems global semaphore set id, 2 is the number of operations.
```

Every chef waits their specific ingredients by this system calls.

Cancelling threads

After wholesaler is done, chefs should not wait ingredients to prepare desert, they must stop their execution. In order to achieve this, I used thread cancellation method. The wholesaler cancels the chef threads after he/she completes delivery.

By using **pthread_cancel()** function, other threads (chefs) can cancelled easily.

- **The pthread_cancel() function sends a cancellation request to the thread thread. Whether and when the target thread reacts to the cancellation request depends on two attributes that are under the control of that thread: its cancelability state and type.**[4]

[3] Linux manual page
[4] Linux manual page

Cancelling chefs in wholesaler (main) thread code snippet ;

```
void _cancel_chefs()
{
	for (int i = 0; i < 6; ++i)
		pthread_cancel(chefs[i]);
}
```

But, target thread must be asynchronous cancelable. Otherwise, default action of thread is deferred cancelable (synchronous). If a thread uses the default action (deferred), it must check the cancel requests by **pthread_testcancel()** explicitly. But in our case, we cannot do that because we can't be sure about semop() or other semaphore functions does this control.

In order to the cancel the target (chef) threads at any given moment, we have to make sure that these threads must **asynchronous cancelable**. (and also cancellation is enabled.)

This operations can be done in the target thread (chef) by this functions ;

int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);

Configuring settings in the chef thread code snippet ;

```
pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS,NULL);   // set as asynchronous
pthread_setcancelstate(PTHREAD_CANCEL_ENABLE,NULL);                // set as enabled
```

Main structure of program

```
_init_semaphores();

_start_chef_threads();

run_wholesaler(filepath);

_cancel_chefs();

/* clear threads */
_join_chef_threads();

/* clear semaphores */
_clear_semaphores();
```

Other implementation notes

In the source code, system functions which are responsible for initializing semaphores, initializing and starting the threads, cancelling the threads, clearing the threads and semaphores are start by underscore (_) character to differentiate with other helper functions in an easy way.

System functions :

- void _init_sigint();
- void _init_semaphores();
- void _clear_semaphores();
- void _start_chef_threads();
- void _join_chef_threads();
- void _cancel_chefs();

Actors (chef and wholesaler) functions :

Chef functions :

- void* run_chef(void* arg); (thread function)
- void print_chef_atwait(int chefno,const char* item1,const char* item2);
- void print_chef_hastaken(int chefno,const char* item1,const char* item2);
- void prepare_desert(int chefno,int sleeptime);
- void print_chef_delivered(int chefno);
- void post_desert_prepared();
- void get_item();

Wholesaler function :

- void run_wholesaler(const char* filepath);
- void print_whslr_deliver(const char* item1,const char* item2);
- void print_whslr_atwait();
- void print_whslr_obtain();
- void post_MF();
- void post_MW();
- void post_MS();
- void post_WS();
- void post_WF();
- void post_FS();
- void wait_desert_prepared();
- void add_item(const char* item_pair);