CS 503 Lab 1

Process Management using Semaphores
The Farmer and The Vegetarians

Shuheng Guo

1. Introduction

In this lab, we are supposed to solve a real problem related to farmers and vegetarians using semaphores. The objective of this lab is as follows:

1) Understand how to create processes in XINU and pass parameters to them

2) Create multiple processes that all need to access a shared buffer

3) Perform process synchronization using semaphores to provide mutual exclusion to a shared resource.

The major requirements and questions are:

- The field has fixed size of FIELDSIZE, so a farmer cannot grow more carrots if the field is full. If this happens the farmer must wait until there is an open spot in the field. Farmers cannot share spots in the field.
- Vegetarians cannot share carrots. If there is only 1 carrot available in a field and 2 vegetarians want to buy and consume it, only one should be allowed to do so.
- How does your solution guarantee that only one process will attempt to remove a carrot at a given time?
- In your solution, when a producer needs to insert a carrot in the queue, are consumers also excluded, or can consumers continue concurrently?
- When a consumer starts to extract carrots, does your system guarantee that the consumer will receive contiguous locations in the queue, or do consumers contend for carrots?

2. Implementation and Explanation

To solve these questions I plan to use semaphore as basic ideas. Using four semaphore with different functionality, two main function representing farmers and vegetarians and one queue realizing by an array acting as circular buffer with functionality of queue. Queue has two operations as enqueue and dequeue which will represent producing and consuming carrots.

Global variables:

```
sid32 general, full, none, mutex; // four semaphore
int32 pid[NFARMERS + NVEGETARIANS]; // record process id
int32 buy[NVEGETARIANS][NFARMERS];
//record how many carrots were bought by each vagetarian
char q[FIELDSIZE]; //cirqular buffer
int32 count = 0; //to count carrots in field
int32 p = 0;
int32 head = 0; // queue head
int32 tail = 0; // queue tail
int32 waitinnone = 0;
int32 waitinfull = 0;
void enqueueue(char); // produce a carrot
void dequeueue(int32); // consume a carrot
void produce(int32);
void consume(int32);
```

Enqueueue and Dequeueue

```
void enqueueue(char item){
        q[tail] = item;
        tail = (tail + 1) % FIELDSIZE; // circular
        count++;
}
void dequeueue(int32 j){
        char item = q[head];
        head = (head + 1) % FIELDSIZE; //circular
        count--;
        int32 i = 0;
```

```
        for(; i < NFARMERS; i++){
            if (item == farmer_tags[i]){
                //record who sell the carrot
                buy[j][i]++;
            }
        }
    }
```

These two functions will be called in function produce and consume respectively representing producing and consuming carrots. Every time a process use dequeueue function, I record who sold the carrot.

Start farm:

```
void start_farm(void)
{
    general = semcreate(0);
    full = semcreate(0);
    none = semcreate(0);
    mutex = semcreate(1);
    int32 i = 0;
    for(; i < NFARMERS; i++, p++){
        pid[p]= create(produce, 4096, 20, farmer_tags[i], 1, i);
        resume(pid[p]);
    }
    int32 j = 0;
    for(; j < NVEGETARIANS; j++, p++){
        pid[p] = create(consume, 4096, 20, vegetarian_tags[j], 1, j);
        resume(pid[p]);
    }
```

```
    signaln(general, NFARMERS+NVEGETARIANS);
}
```

In this lab Implementation, I created four semaphore in total, general, mutex, full, none, respectively. Their functions are as follows:

1) general: Every process is going to wait in general at first and be started at the same time by function signaln.

2)Mutex: To realize mutual exclusive oepration of queue.

3)Full: Farmers will wait in semaphore full when the field is already full.

4)None: Vegetarians will wait in semaphore none when the carrots in the field are less than vegetarians' hunger index.

Produce:

```
void produce(int32 i){
        wait(general);
        while(TRUE){
            if( count >= FIELDSIZE){
                waitinfull++;
                wait(full);
            }
            else {
                sleepms(farmer_grow_times[i]);
                wait(mutex);
                enqueueue(farmer_tags[i]);
                if(waitinnone > 0){
                    signal(none);
                    waitinnone--;
                }
                signal(mutex);
```

```
        }
    }
}
```

Function produce is used by farmer to produce carrots. First wait in general to start all process at the same time. If the field is full, farmer will be waiting in semaphore full. If that is not the case, all the farmer will sleep at the beginning to ensure that vegetarian consume the ripe carrots. After producing a carrot check if there is any process waiting in the semaphore none and signal it if there is one.

Consume:

```
void consume(int32 j){
        wait(general);
        while(TRUE){
            wait(mutex);
            if(count <= vegetarian_hungers[j]){
                waitinnone++;
                signal(mutex);
                wait(none);
                continue;
            }
            else{
                int32 k = 0;
                for(; k < vegetarian_hungers[j]; k++){
                    dequeueue(j);
                    if(waitinfull > 0){
                        signal(full);
                        waitinfull--;
                    }
```

```
                    }
                    signal(mutex);
                    sleepms(vegetarian_sleep_times[j]);
                }
            }
        }
```

Function consume is used by vegetarian to consume carrots. First wait in general to start all process at the same time. If the number of carrots in the field is less then vegetarians' hunger-index, the vegetarian will be waiting in semaphore none. If that is not the case, vegetarians will buy and consume carrots then check if there is any process waiting in the semaphore full and signal it if there is one.

Stop farm:

```
void stop_farm(void)
{
    int32 i = 0;
    for(; i < NFARMERS + NVEGETARIANS; i++){
        kill(pid[p]);
    }
}
```

Stop farm is used to kill all the processes after 30 seconds.

Print Report:

```
void print_farm_report(void)
{
    int32 i = 0;
    for(; i < NFARMERS; i++){
        int sum = 0;
        int u = 0;
```

```
            for(; u < NVEGETARIANS; u++){
                sum = sum + buy[u][i];
            }
            kprintf("Farmer %c : sold %d carrots\n", farmer_tags[i], sum);
        }
        int32 j = 0;
        for(; j < NVEGETARIANS; j++){
            kprintf("vegetarian %c :", vegetarian_tags[j]);
            int32 k = 0;
            for(; k < NFARMERS; k++){
                kprintf(" %d carrots from farmer %c,", buy[j][k], farmer_tags[k]);
            }
            kprintf("\n");
        }
}
```

Print the lab result according to the format in lab description.

To answer these questions:

- Q: How does your solution guarantee that only one process will attempt to remove a carrot at a given time?
- A: I used a semaphore called mutex to prevent multiple operations on Queue at a given time, ie processes with queue operations are mutual exclusive.
- Q: In your solution, when a producer needs to insert a carrot in the queue, are consumers also excluded, or can consumers continue concurrently?
- A: In my solution, when a producer is going to insert a carrot in the queue, all consumers are excluded by waiting on the mutex semaphore. Because the queue operation is mutual exclusive.
- Q: When a consumer starts to extract carrots, does your system guarantee that the

consumer will receive contiguous locations in the queue, or do consumers contend for carrots?

- A: Yes, I can guarantee that the consumers will receive contiguous locations in the queue. Because the dequeue is mutual exclusive. If the carrots in queue are less than the number of carrots every vegetarian needs to buy, the vegetarian process will wait in a semaphore called none.
- Q :The field has fixed size of FIELDSIZE, so a farmer cannot grow more carrots if the field is full. If this happens the farmer must wait until there is an open spot in the field. Farmers cannot share spots in the field.
- A: This requirement is realized by a semaphore called full. Whenever the field is full, a farmer cannot grow carrots and must wait in the full semaphore until there is an open spot in the field.
- Q:Vegetarians cannot share carrots. If there is only 1 carrot available in a field and 2 vegetarians want to buy and consume it, only one should be allowed to do so.
- A: Since I use semaphore mutex before the if conditioning sentence and every vegetarian who awakes from semaphore will continue the while loop and judge the if conditioning again, only one vegetarian can buy and consume carrot.

3. Results

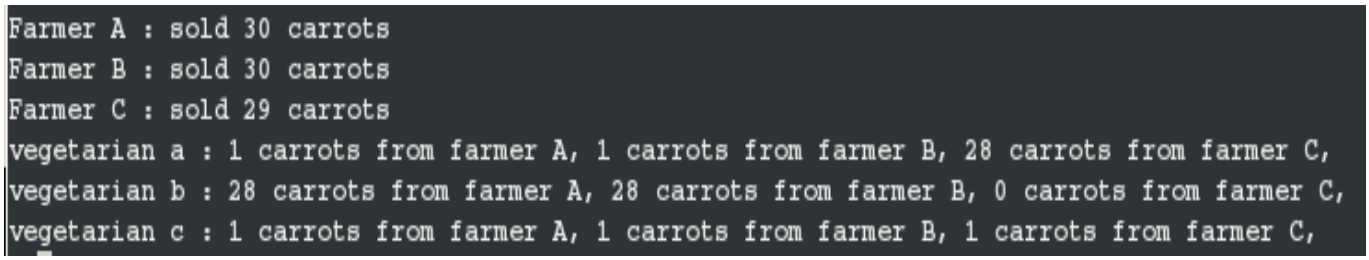

Figure 1. Result from equal priority

From Figure 1 we can see that Farmer A, B and C produced almost the same number of carrots. This is reasonable because the total processing duration is about 30 seconds and every farmer sleep 1 second before producing a carrot.

Vegetarian a bought 1 carrot from A, 1 carrot from B and 28 carrots from C. Vegetarian

b brought 28 carrots from A, 28 carrots from B, and 0 carrot from C. Vegetarian c brought 1 carrot from A, 1 carrot from B, and 1 carrot from C.

This result is reasonable because in farm.h, we define that vegetarian b will buy 2 carrots every time and vegetarian a, c will buy 1 carrot each time. That means that when there are not enough carrots in the field, b will buy all the carrots and leave c few opportunities to buy a carrot.

4. Extra Credits Experiments

4.1. The farmers have higher priority than the vegetarians

```
Farmer A : sold 29 carrots
Farmer B : sold 29 carrots
Farmer C : sold 29 carrots
vegetarian a : 1 carrots from farmer A, 15 carrots from farmer B, 14 carrots from farmer C,
vegetarian b : 14 carrots from farmer A, 7 carrots from farmer B, 7 carrots from farmer C,
vegetarian c : 15 carrots from farmer A, 7 carrots from farmer B, 8 carrots from farmer C,
```

Figure 2. Result from farmer high priority

I changed farmer's priority to 30 and vegetarian's priority remain 20 unchanged.

From Figure 2 we can see that Farmer A, B and C produced almost the same number of carrots. This is reasonable because the total processing duration is about 30 seconds and every farmer sleep 1 second before producing a carrot.

Vegetarian a bought 1 carrot from A, 15 carrots from B and 14 carrots from C. Vegetarian b brought 14 carrots from A, 7 carrots from B, and 7 carrots from C. Vegetarian c brought 15 carrots from A, 7 carrots from B, and 8 carrots from C.

4.2. The vegetarians have higher priority than the farmers

```
Farmer A : sold 29 carrots
Farmer B : sold 29 carrots
Farmer C : sold 28 carrots
vegetarian a : 5 carrots from farmer A, 5 carrots from farmer B, 15 carrots from farmer C,
vegetarian b : 20 carrots from farmer A, 20 carrots from farmer B, 0 carrots from farmer C,
vegetarian c : 5 carrots from farmer A, 5 carrots from farmer B, 14 carrots from farmer C,
```

Figure 3. Result from vegetarian higher priority

I changed vegetarian's priority to 30 and farmer's priority remain 20 unchanged.

From Figure 3 we can see that Farmer A, B and C produced almost the same number of carrots. This is reasonable because the total processing duration is about 30 seconds and every farmer sleep 1 second before producing a carrot.

Vegetarian a bought 5 carrots from A, 5 carrots from B and 15 carrots from C. Vegetarian b brought 20 carrots from A, 20 carrots from B, and 0 carrot from C. Vegetarian c brought 5 carrots from A, 5 carrots from B, and 14 carrots from C.

4.3. Each vegetarian to have a unique priority (high, medium, low)

```
Farmer A : sold 29 carrots
Farmer B : sold 29 carrots
Farmer C : sold 29 carrots
vegetarian a : 14 carrots from farmer A, 0 carrots from farmer B, 14 carrots from farmer C,
vegetarian b : 15 carrots from farmer A, 15 carrots from farmer B, 0 carrots from farmer C,
vegetarian c : 0 carrots from farmer A, 14 carrots from farmer B, 15 carrots from farmer C,
```

Figure 4. Result from vegetarian unique priority

I changed vegetarian's priority to unique priorities and farmer's priority remain 20 unchanged.

From Figure 4 we can see that Farmer A, B and C produced almost the same number of carrots. This is reasonable because the total processing duration is about 30 seconds and every farmer sleep 1 second before producing a carrot.

Vegetarian a bought 14 carrots from A, 0 carrots from B and 14 carrots from C. Vegetarian b brought 15 carrots from A, 15 carrots from B, and 0 carrot from C. Vegetarian c brought 0 carrots from A, 14 carrots from B, and 15 carrots from C.

Comparing these results from different priorities, we can conclude that the results from vegetarian with higher priority provide a better fairness. Since the number of carrots each vegetarian bought from farmers is proportional to everyone's hunger index. (In this case: 1:2:1)