Ayoola Etiko, cs3413, 3598133

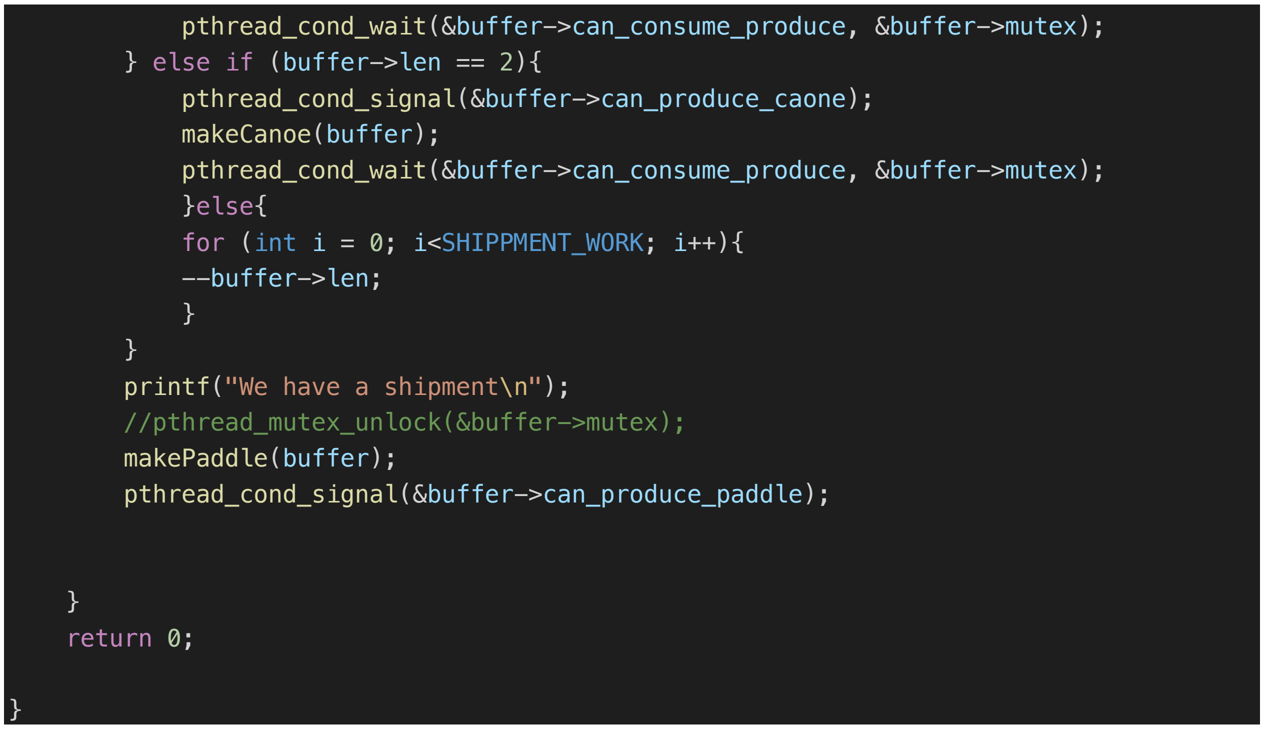
```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#define BUFFER_SIZE 3
#define SHIPPMENT_WORK 3

typedef struct salesPackage
{
    int bufferArr[BUFFER_SIZE]; //number of items in the buffer
    size_t len; //number of items in buffer
    pthread_mutex_t mutex; // needed to add/remove data from the buffer
    pthread_cond_t can_produce_caone; //signaled when are removed from my buffer
    pthread_cond_t can_produce_paddle; //signaled when are removed from my buffer
    pthread_cond_t can_consume_produce; //signaled when are added from my buffer

}salesPackage;

void * makeCanoe(void * arg);
void * makePaddle(void * arg);
void * makeShipment(void * arg);


int main(int argc, char *argv[]){
    salesPackage buffer = {
        .len = 0,
        .mutex = PTHREAD_MUTEX_INITIALIZER,
        .can_produce_caone = PTHREAD_COND_INITIALIZER,
        .can_produce_paddle = PTHREAD_COND_INITIALIZER,
        .can_consume_produce = PTHREAD_COND_INITIALIZER
    };

    int n = atoi(argv[1]);
    time_t t =  (time_t) n;
    srand((unsigned) time(&t));
    //generating random numbers between 2 and 5 inclusive

    int randomTimeForThread = (rand() %4) + 2;
    //printf("%d\n", randomTimeForThread);
    pthread_t canoeThread;
    pthread_t paddleThread;
    pthread_t shipperThread;
    int x;

    for(x = n; x >= 0; x = x-randomTimeForThread){
        pthread_create (&paddleThread, NULL, &makePaddle, (void*)&buffer);
        //this will let the canoe thread wait for the canoe thread to be done
        pthread_join(paddleThread, NULL);
```

Ayoola Etiko, cs3413, 3598133

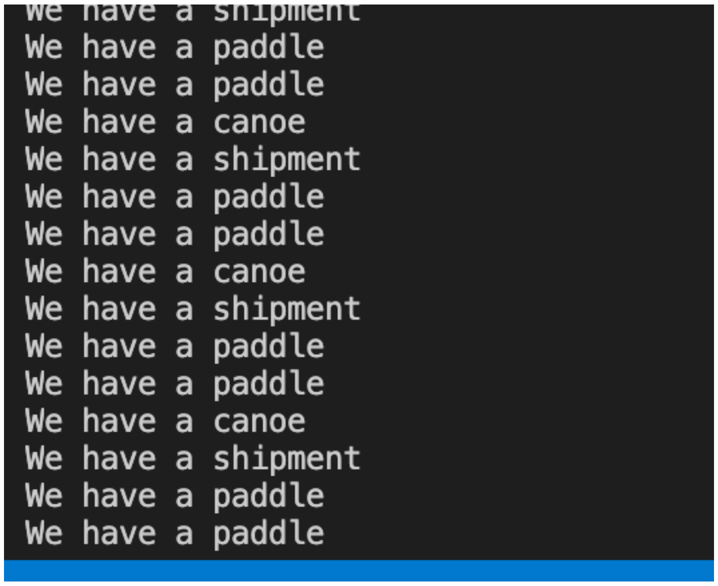
```
        pthread_create (&canoeThread, NULL, &makeCanoe, (void*)&buffer);
        //this will let the paddle thread wait for the canoe thread to be done
        pthread_join(canoeThread, NULL);
        pthread_create (&shipperThread, NULL, &makeShipment,(void*)&buffer);
        pthread_join(shipperThread, NULL);

    }
    if(x == 0){
            pthread_detach(canoeThread);
            pthread_detach(paddleThread);
            pthread_detach(shipperThread);

        }
    pthread_exit(NULL);

    return 0;
}

void * makePaddle(void * arg){
    salesPackage *buffer = (salesPackage*)arg;
        while(1){
            #ifdef UNDERFLOW
            // used to show that if the producer is somewhat "slow" the consumer will
not fail (i.e. it'll just wait for new items to consume)
            sleep(rand() % 3);
            #endif

            pthread_mutex_lock(&buffer->mutex);
            if(buffer->len == BUFFER_SIZE){
                //if full wait till all is consumed
                pthread_cond_signal(&buffer->can_consume_produce);
                makeShipment(buffer);
                pthread_cond_wait(&buffer->can_produce_paddle, &buffer->mutex);
                pthread_mutex_unlock(&buffer->mutex);
                //if buffer is size 0 or 1 produce paddles
            } else if(buffer->len == 2){
                pthread_cond_signal(&buffer->can_produce_caone);
                makeCanoe(buffer);
                pthread_cond_wait(&buffer->can_produce_paddle, &buffer->mutex);
            }
            printf("We have a paddle\n");
            buffer->bufferArr[buffer->len] = 1;
            buffer->len++;
            pthread_mutex_unlock(&buffer->mutex);
        }

        return 0;
```

```
}


void * makeCanoe(void * arg){
    salesPackage *buffer = (salesPackage*)arg;
    while(1){
        #ifdef UNDERFLOW
        // used to show that if the producer is somewhat "slow" the consumer will not
fail (i.e. it'll just wait for new items to consume)
        sleep(rand() % 3);
        #endif
        pthread_mutex_unlock(&buffer->mutex);
        printf("We have a canoe\n");
        buffer->bufferArr[buffer->len] = 1;
        buffer->len++;


        if(buffer->len == BUFFER_SIZE){
            //if full wait till all is consumed
            pthread_cond_signal(&buffer->can_consume_produce);
            makeShipment(buffer);
            pthread_cond_wait(&buffer->can_produce_caone, &buffer->mutex);
            pthread_mutex_unlock(&buffer->mutex);
            //if buffer is size 0 or 1 produce paddles
        } else if(buffer->len == 1 || buffer->len == 0){
            pthread_cond_signal(&buffer->can_produce_paddle);
            makePaddle(buffer);
            pthread_cond_wait(&buffer->can_produce_caone, &buffer->mutex);
        }
    }
    //pthread_mutex_unlock(&buffer->mutex);
    return 0;
}


void * makeShipment(void * arg){
    salesPackage *buffer = (salesPackage*)arg;
    while(1){
        #ifdef OVERFLOW
        // showing that the buffer won't overflow if the consumer is slow (i.e. the
producer will wait)
        sleep(rand() % 3);
        #endif
        pthread_mutex_unlock(&buffer->mutex);

        if(buffer->len == 0|| buffer->len == 1){
            pthread_cond_signal(&buffer->can_produce_paddle);
            makePaddle(buffer);
```

Ayoola Etiko, cs3413, 3598133



```
            pthread_cond_wait(&buffer->can_consume_produce, &buffer->mutex);
        } else if (buffer->len == 2){
            pthread_cond_signal(&buffer->can_produce_caone);
            makeCanoe(buffer);
            pthread_cond_wait(&buffer->can_consume_produce, &buffer->mutex);
            }else{
            for (int i = 0; i<SHIPPMENT_WORK; i++){
            --buffer->len;
            }
        }
        printf("We have a shipment\n");
        //pthread_mutex_unlock(&buffer->mutex);
        makePaddle(buffer);
        pthread_cond_signal(&buffer->can_produce_paddle);


    }
    return 0;

}
```



```
We have a shipment
We have a paddle
We have a paddle
We have a canoe
We have a shipment
We have a paddle
We have a paddle
We have a canoe
We have a shipment
We have a paddle
We have a paddle
We have a canoe
We have a shipment
We have a paddle
We have a paddle
```