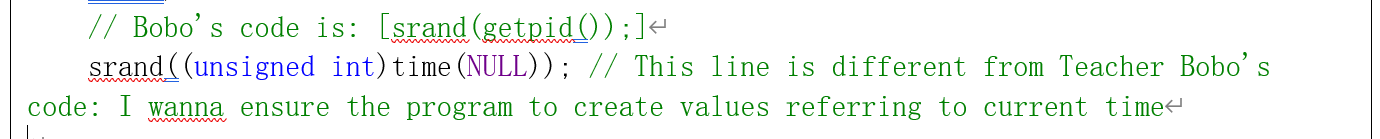
# Operating System Lab\_05

22CST 蒋云翔 2022102330

## Task1：

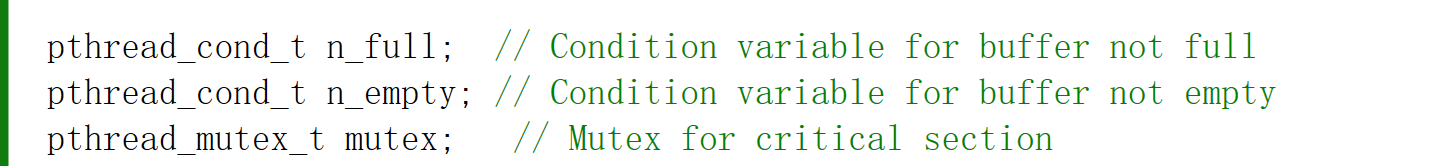
### Idea：



**The overall design idea of this program：**

* **Thread Creation**: Create a specified number of producer and consumer threads.
* **Data Sharing**: Define a shared buffer where producers can add data and consumers can remove data.
* **Synchronization Mechanism**: Use mutexes and condition variables for thread synchronization.

**The key points of this program and the key ideas to solve this problem:**





**1. Mutex (Mutual Exclusion)**

* **Purpose**: Protect access to shared resources (buffer) to prevent inconsistencies due to simultaneous access by multiple threads.
* **Implementation**: Use pthread\_mutex\_t to initialize a mutex and pthread\_mutex\_lock and pthread\_mutex\_unlock to lock and unlock the buffer access.

**2. Condition Variables**

* **Purpose**: Control the execution timing of producers and consumers, making producers wait when the buffer is full and consumers wait when the buffer is empty.
* **Implementation**: Use pthread\_cond\_t to initialize condition variables and combine them with mutexes using pthread\_cond\_wait and pthread\_cond\_signal for waiting and signaling mechanisms.

**3. Buffer Management**

* **Data Structure**: Design a fixed-size array as the buffer.
* **Index Management**: Use indices (e.g., in\_index and out\_index) to track where producers can add data and where consumers can remove data.

**4. Thread Creation and Synchronization**

* **Thread Identification**: Assign a unique identifier (ID) to each thread and pass it to the thread function.
* **Thread Functions**: Write pro\_route and con\_route functions to implement the behaviors of producers and consumers, respectively.

## Code structure

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <pthread.h>  #include <string.h>  #define PRO\_NUM 3 // Number of producer threads  #define CON\_NUM 2 // Number of consumer threads  #define BUFFER\_SIZE 3 // Size of the buffer (Ps:can be changed!!!To simulate Bobo's expected execution, I set 3 here!!!)  pthread\_cond\_t n\_full; // Condition variable for buffer not full  pthread\_cond\_t n\_empty; // Condition variable for buffer not empty  pthread\_mutex\_t mutex; // Mutex for critical section  int buffer[BUFFER\_SIZE];  int in\_index = 0; // Index to insert into the buffer  int out\_index = 0; // Index to remove from the buffer  int g\_num = 0; // Number of products in the buffer  // Producer thread function  void\* pro\_route(void\* arg) {  int id = \*((int\*)arg);  while (1) {  pthread\_mutex\_lock(&mutex);  while (g\_num == BUFFER\_SIZE) {  printf("Producer thread %d must wait.\n", id);  pthread\_cond\_wait(&n\_empty, &mutex); // Buffer is full, wait  }  // Produce an item  int item = rand() % 100 + 1; // Random item to produce  buffer[in\_index] = item;  in\_index = (in\_index + 1) % BUFFER\_SIZE;  g\_num++;  printf("Producer thread %d now produces\n", id);  printf("The number of products produced %d\n", g\_num);  pthread\_cond\_signal(&n\_full); // Signal that buffer is not full  pthread\_mutex\_unlock(&mutex);  // No need to free(arg) here, as we are using a loop  sleep(1); // Simulate time taken to produce an item  }  return NULL;  }  // Consumer thread function  void\* con\_route(void\* arg) {  int id = \*((int\*)arg);  while (1) {  pthread\_mutex\_lock(&mutex);  while (g\_num == 0) {  printf("Consumer thread %d must wait.\n", id);  pthread\_cond\_wait(&n\_full, &mutex); // Buffer is empty, wait  }  // Consume an item  int item = buffer[out\_index];  out\_index = (out\_index + 1) % BUFFER\_SIZE;  g\_num--;  printf("The consumer thread %d consumes product %d\n", id, item);  printf("The consumer thread %d has finished consuming product\n", id);  pthread\_cond\_signal(&n\_empty); // Signal that buffer is not full  pthread\_mutex\_unlock(&mutex);  sleep(1); // Simulate time taken to consume an item  }  return NULL;  }  int main() {  // Bobo's code is: [srand(getpid());]  srand((unsigned int)time(NULL)); // This line is different from Teacher Bobo's code: I wanna ensure the program to create values referring to current time  pthread\_t tids[PRO\_NUM + CON\_NUM];  // Initialize mutex and condition variables  pthread\_mutex\_init(&mutex, NULL);  pthread\_cond\_init(&n\_full, NULL);  pthread\_cond\_init(&n\_empty, NULL);  // Create producer threads  for (int i = 0; i < PRO\_NUM; ++i) {  int\* p = (int\*)malloc(sizeof(int));  \*p = i;  pthread\_create(&tids[i], NULL, pro\_route, p);  }  // Create consumer threads  for (int i = 0; i < CON\_NUM; ++i) {  int\* p = (int\*)malloc(sizeof(int));  \*p = i;  pthread\_create(&tids[i + PRO\_NUM], NULL, con\_route, p);  }  // Wait for all threads to finish (this will never happen in this loop)  for (int i = 0; i < PRO\_NUM + CON\_NUM; ++i) {  pthread\_join(tids[i], NULL);  }  // Destroy mutex and condition variables  pthread\_mutex\_destroy(&mutex);  pthread\_cond\_destroy(&n\_full);  pthread\_cond\_destroy(&n\_empty);  return 0;  } |

## Execution results:

|  |
| --- |
| **Stop the program by ctrl+z** |