# Operating Systems – Project 1

- Simple Scheduling -

ChangYoon Lee, ChanHo Park

Dankook University, Korea Republic
32183641@gmail.com, chanho.park@dankook.ac.kr

**Abstract.** Most modern operating systems have extended the concept of the process to allow having multiple threads of execution and thus to perform more than one task at a time (*Silberschatz et al., 2014*). While allowing this multi-threaded concept, there are some issues to consider in designing it: Synchronization (Concurrency Control). In this paper, we will discuss the concept of the multi-thread, synchronization examples, and their solutions, especially on producer and consumer problems. Also, we will implement the example case of producer and consumer problems with a multi-threaded word count program and evaluate it.

**Keywords:** Concurrency Control, Multi-Thread, Mutex, Producer and Consumer, Semaphore, Synchronization, Reader and Write

## 1 Introduction

Thread is a unit of execution. It has an execution context, which includes the registers, and stack. Denote that the address space in the memory is shared among the threads in the same process, so there is no clear separation and protection for accessing the memory space among the threads in the same process (Yoo, Mobile-os-DKU-cis-MSE). This single thread allows the process to perform only one task simultaneously. However, modern operating systems support the process of having multiple threads, so that they can execute multiple tasks parallelly at a time.

The concept of multi-threaded programming has some benefits, but there are some problems to be resolved to apply the following concepts, such as synchronization, mutual exclusion, deadlock, starvation, and optimization. To determine the following problem, we use several solutions such as queue, mutex, semaphore, and optimization methods, for the synchronization.

In this paper, we will first explain the concepts of thread, multi-thread, problems along the multi-threaded programming, and their solutions. By applying these concepts, we will explain how we implemented the multi-threaded word count program and its results for versions 1 through 3. Also, we will show the optimization with some methods to present the enhanced performance. At the end of the paper, we will present the execution time among the differences between the number of threads and implementation methods.

# 2 Requirements

| Index | Requirement |
|---|---|
| 1 | Correct the values in the thread and keep the consistencies in the given producer and consumer program. |
| 2 | Correct the given code for producer and consumer program so that it works with single producer and single consumer. |
| 3 | Enhance the given producer and consumer program to support multiple consumers. |
| 4 | Make consumer threads to gather some statistics of the given text in the file.<br>a. Count the number of each alphabet character in the line.<br>b. At the end of the execution, the program should print out the statistic of the entire text.<br>c. Reach to the fastest execution and maximize the concurrency. |

**Figure 1 - Requirement Specification**

Figure 1 shows the requirements for a multi-threaded word count program. The implementations for these requirements will be described in detail afterwards.

# 3 Concepts

## 3.1 Process

The normal process model implies that a process is a program that performs a single thread of execution. For example, when a process is running a word-processor program, a single thread of the instructions is being executed. This single thread of control allows the process to perform only one task at a time (*Silberschatz et al., 2014*). On the systems that supports thread, the process control block (PCB) is expanded to include the information for the thread. Other changes throughout the system are also needed to support the threads.

### 3.1.2 Process State

The normal process model implies that a process is a program that performs a single thread of execution. For example, when a process is running a word-processor program, a single thread of the instructions is being executed. This single thread of control allows the process to perform only one task at a time (*Silberschatz et al., 2014*). On the systems that supports thread, the process control block (PCB) is expanded to include the information for the thread.

### 3.1.3 Process Control Block

The normal process model implies that a process is a program that performs a single thread of execution. For example, when a process is running a word-processor program, a single thread of the instructions is being executed.

### 3.2 Process Scheduling

In this paper, we will focus on applying the concept of multi-threaded programming, and the solutions for resolving the presented challenges in the word count program. These challenges will also be described in detail, in a section on the optimization of the implemented program.

#### 3.2.1 Scheduling Queues

In this paper, we will focus on applying the concept of multi-threaded programming, and the solutions for resolving the presented challenges in the word count program. These challenges will also be described in detail, in a section on the optimization of the implemented program.

#### 3.2.2 Scheduler

In this paper, we will focus on applying the concept of multi-threaded programming, and the solutions for resolving the presented challenges in the word count program. These challenges will also be described in detail, in a section on the optimization of the implemented program.

### 3.3 Inter-Process Communication (IPC)

There are several solutions for the following situation that satisfies the requirements above. In this paper, we will present two main solutions for resolving the critical-section problem, which is mutex and semaphore. The details of the mutex and semaphore will be presented in the later sections.

#### 3.3.1 Message Passing

There are several solutions for the following situation that satisfies the requirements above. In this paper, we will present two main solutions for resolving the critical-section problem, which is mutex and semaphore. The details of the mutex and semaphore will be presented in the later sections.

#### 3.3.2 Message Queue Naming

There are several solutions for the following situation that satisfies the requirements above. In this paper, we will present two main solutions for resolving the critical-section problem.

### 3.3.3 Message Queue Synchronization

There are several solutions for the following situation that satisfies the requirements above. In this paper, we will present two main solutions for resolving the critical-section problem, which is mutex and semaphore. The details of the mutex and semaphore will be presented in the later sections.

### 3.3.4 Message Queue Buffering

There are several solutions for the following situation that satisfies the requirements above. In this paper, we will present two main solutions for resolving the critical-section problem, which is mutex and semaphore. The details of the mutex and semaphore will be presented in the later sections.

## 3.4 CPU Scheduling

In this paper, we will focus on applying the concept of multi-threaded programming, and the solutions for resolving the presented challenges in the word count program. These challenges will also be described in detail, in a section on the optimization of the implemented program.

### 3.4.1 Scheduling Criteria

In this paper, we will focus on applying the concept of multi-threaded programming, and the solutions for resolving the presented challenges in the word count program. These challenges will also be described in detail, in a section on the optimization of the implemented program.

### 3.4.2 Scheduling Algorithms

In this paper, we will focus on applying the concept of multi-threaded programming, and the solutions for resolving the presented challenges in the word count program. These challenges will also be described in detail, in a section on the optimization of the implemented program.

### 3.4.3 Algorithm Evaluation

In this paper, we will focus on applying the concept of multi-threaded programming, and the solutions for resolving the presented challenges in the word count program.

# 4 Simple Scheduling

## 4.1 Signal and Handler

Figure 4 shows the solution code that uses mutex lock to resolve the critical section problem. The acquire function acquires the lock, and the release function releases the lock. A mutex has a Boolean variable whose value indicates whether the lock is available or not. If the lock is available, the acquire function succeeds, and the lock is then considered to be unavailable. A thread that attempts to acquire an unavailable lock is blocked until the lock is released.
The main disadvantage of the implementation of the mutex lock is that it acquires busy waiting. While a process or thread is in its critical section, any other process that tries to enter its critical section must loop continuously in the call of acquiring function. This continual looping becomes a problem in the multi-programming system because it wastes the CPU cycle that some other processes and threads might be able to use productively. In the aspect of continuously looping for the busy wait, the mutex lock is also called a spinlock because the thread spins while waiting for the lock to become available.

## 4.2 Message Passing in POSIX

Figure 4 shows the solution code that uses mutex lock to resolve the critical section problem. The acquire function acquires the lock, and the release function releases the lock. A mutex has a Boolean variable whose value indicates whether the lock is available or not. If the lock is available, the acquire function succeeds, and the lock is then considered to be unavailable. A thread that attempts to acquire an unavailable lock is blocked until the lock is released.
The main disadvantage of the implementation of the mutex lock is that it acquires busy waiting. While a process or thread is in its critical section, any other process that tries to enter its critical section must loop continuously in the call of acquiring function. This continual looping becomes a problem in the multi-programming system because it wastes the CPU cycle that some other processes and threads might be able to use productively. In the aspect of continuously looping for the busy wait, the mutex lock is also called a spinlock because the thread spins while waiting for the lock to become available.

## 4.3 First-Come First-Served Algorithm (FCFS)

Figure 4 shows the solution code that uses mutex lock to resolve the critical section problem. The acquire function acquires the lock, and the release function releases the lock. A mutex has a Boolean variable whose value indicates whether the lock is available or not. If the lock is available, the acquire function succeeds, and the lock is then considered to be unavailable. A thread that attempts to acquire an unavailable lock is blocked until the lock is released.
The main disadvantage of the implementation of the mutex lock is that it acquires busy waiting. While a process or thread is in its critical section, any other process that tries to

enter its critical section must loop continuously in the call of acquiring function. This continual looping becomes a problem in the multi-programming system because it wastes the CPU cycle that some other processes and threads might be able to use productively. In the aspect of continuously looping for the busy wait, the mutex lock is also called a spinlock because the thread spins while waiting for the lock to become available.

### 4.4 Shortest Job First Algorithm (SJF)

Figure 4 shows the solution code that uses mutex lock to resolve the critical section problem. The acquire function acquires the lock, and the release function releases the lock. A mutex has a Boolean variable whose value indicates whether the lock is available or not. If the lock is available, the acquire function succeeds, and the lock is then considered to be unavailable. A thread that attempts to acquire an unavailable lock is blocked until the lock is released.
The main disadvantage of the implementation of the mutex lock is that it acquires busy waiting. While a process or thread is in its critical section, any other process that tries to enter its critical section must loop continuously in the call of acquiring function. This continual looping becomes a problem in the multi-programming system because it wastes the CPU cycle that some other processes and threads might be able to use productively. In the aspect of continuously looping for the busy wait, the mutex lock is also called a spinlock because the thread spins while waiting for the lock to become available.

### 4.5 Round Robin Algorithm (RR)

Figure 4 shows the solution code that uses mutex lock to resolve the critical section problem. The acquire function acquires the lock, and the release function releases the lock. A mutex has a Boolean variable whose value indicates whether the lock is available or not. If the lock is available, the acquire function succeeds, and the lock is then considered to be unavailable. A thread that attempts to acquire an unavailable lock is blocked until the lock is released.
The main disadvantage of the implementation of the mutex lock is that it acquires busy waiting. While a process or thread is in its critical section, any other process that tries to enter its critical section must loop continuously in the call of acquiring function. This continual looping becomes a problem in the multi-programming system because it wastes the CPU cycle that some other processes and threads might be able to use productively. In the aspect of continuously looping for the busy wait, the mutex lock is also called a spinlock because the thread spins while waiting for the lock to become available.

### 4.6 Completely Fair Safe Algorithm (CFS)

Figure 4 shows the solution code that uses mutex lock to resolve the critical section problem. The acquire function acquires the lock, and the release function releases the lock. A mutex has a Boolean variable whose value indicates whether the lock is available or not. If the lock

is available, the acquire function succeeds, and the lock is then considered to be unavailable. A thread that attempts to acquire an unavailable lock is blocked until the lock is released.
The main disadvantage of the implementation of the mutex lock is that it acquires busy waiting. While a process or thread is in its critical section, any other process that tries to enter its critical section must loop continuously in the call of acquiring function. This continual looping becomes a problem in the multi-programming system because it wastes the CPU cycle that some other processes and threads might be able to use productively. In the aspect of continuously looping for the busy wait, the mutex lock is also called a spinlock because the thread spins while waiting for the lock to become available.

### 4.7 Program Definition

Before implementing the multi-threaded word count program, we will state the additional program definition that will be used in the real implementation.

- Global Variables

| Variables | Data Type | Definition |
|---|---|---|
| ASCII_SIZE | 256 | Size of the total number of ASCII characters |
| BUFFER_SIZE | 100 | The size of the shared buffer |
| MAX_STRING-LENFTG | 30 | Maximum length of the single word that the word count program will read |

- Modules and Functions

| Modules | Functions | Definition |
|---|---|---|
| prod_cons | producer | Reads the lines from a given file, and put the line string on the shared buffer |
| | consumer | Get string from the shared buffer, and print the line out on the console screen |
| | main | Main thread which performs the admin job |

## 5 Implementation

### 5.1 Queue Header

Figure 4 shows the solution code that uses mutex lock to resolve the critical section problem. The acquire function acquires the lock, and the release function releases the lock. A mutex has a Boolean variable whose value indicates whether the lock is available or not. If the lock is available, the acquire function succeeds, and the lock is then considered to be unavailable. A thread that attempts to acquire an unavailable lock is blocked until the lock is released.
The main disadvantage of the implementation of the mutex lock is that it acquires busy waiting. While a process or thread is in its critical section, any other process that tries to enter its critical section must loop continuously in the call of acquiring function. This continual looping becomes a problem in the multi-programming system because it wastes

the CPU cycle that some other processes and threads might be able to use productively. In the aspect of continuously looping for the busy wait, the mutex lock is also called a spinlock because the thread spins while waiting for the lock to become available.

### 5.2 Heap Header

Figure 4 shows the solution code that uses mutex lock to resolve the critical section problem. The acquire function acquires the lock, and the release function releases the lock. A mutex has a Boolean variable whose value indicates whether the lock is available or not. If the lock is available, the acquire function succeeds, and the lock is then considered to be unavailable. A thread that attempts to acquire an unavailable lock is blocked until the lock is released.
The main disadvantage of the implementation of the mutex lock is that it acquires busy waiting. While a process or thread is in its critical section, any other process that tries to enter its critical section must loop continuously in the call of acquiring function. This continual looping becomes a problem in the multi-programming system because it wastes the CPU cycle that some other processes and threads might be able to use productively. In the aspect of continuously looping for the busy wait, the mutex lock is also called a spinlock because the thread spins while waiting for the lock to become available.

### 5.3 Inter-Process Control (IPC) Message Passing Header

Figure 4 shows the solution code that uses mutex lock to resolve the critical section problem. The acquire function acquires the lock, and the release function releases the lock. A mutex has a Boolean variable whose value indicates whether the lock is available or not. If the lock is available, the acquire function succeeds, and the lock is then considered to be unavailable. A thread that attempts to acquire an unavailable lock is blocked until the lock is released.
The main disadvantage of the implementation of the mutex lock is that it acquires busy waiting. While a process or thread is in its critical section, any other process that tries to enter its critical section must loop continuously in the call of acquiring function. This continual looping becomes a problem in the multi-programming system because it wastes the CPU cycle that some other processes and threads might be able to use productively. In the aspect of continuously looping for the busy wait, the mutex lock is also called a spinlock because the thread spins while waiting for the lock to become available.

### 5.4 Main

Figure 4 shows the solution code that uses mutex lock to resolve the critical section problem. The acquire function acquires the lock, and the release function releases the lock. A mutex has a Boolean variable whose value indicates whether the lock is available or not. If the lock is available, the acquire function succeeds, and the lock is then considered to be unavailable. A thread that attempts to acquire an unavailable lock is blocked until the lock is released.

The main disadvantage of the implementation of the mutex lock is that it acquires busy waiting. While a process or thread is in its critical section, any other process that tries to enter its critical section must loop continuously in the call of acquiring function. This continual looping becomes a problem in the multi-programming system because it wastes the CPU cycle that some other processes and threads might be able to use productively.

# 6 Build Environment

- Build Environment:
  1. Linux Environment -> Vi editor, GCC Complier
  2. Program is built by using the Makefile.

- Build Command:
  1. $make prod_cons -> build the execution program for prod_cons from version 1 to 4.
  2. $make clean -> clean all the object files that consists of the prod_cons programs.
- Execution Command:
  1. . /Prod_cons_v1 {$readfile}
     -> Execute the producer and consumer version 1 program.
  2. . /Prod_cons_v2.1 {$readfile} #Producer #Consumer
     -> Execute the producer and consumer version 2.1 program.
  3. . /Prod_cons_v2.2 {$readfile} #Producer #Consumer
     -> Execute the producer and consumer version 2.2 program.
  4. . /Prod_cons_v2.3 {$readfile} #Producer #Consumer
     -> Execute the producer and consumer version 2.3 program.
  5. . /Prod_cons_v3 {$readfile} #Producer #Consumer
     -> Execute the producer and consumer version 3 program.
  6. . /Prod_cons_v4 {$readfile} #Producer #Consumer
     -> Execute the producer and consumer version 4 program.

# 7 Results

- Producer and Consumer Version 1 reading LICENSE file.

# 8 Evaluation

Figures 35 and 36 show the graph result of execution time per number of the threads for the producer and consumer program versions 2.3 that reads the file of FeeBSD9-Orig.tar and the android.tar. The following program is implemented by applying the methods that are presented previously. As we can see from the figures, the execution time decreases as the

number of threads increases. In short, the producer and consumer program version 2.3 show the ideal and faster result of execution time among the other programs.

## 9 Conclusion

By understanding this paper, we can understand the basic concepts of thread and multi-threaded programming. Also, we can understand the problems and the solutions that occur by applying the following concept, which is about data dependency and synchronization.

## Citations


[1] Kirvan, P. (2022, May 26). *What is multithreading?* WhatIs.com. Retrieved September 11, 2022, from https://www.techtarget.com/whatis/definition/multithreading

[2] Silberschatz, A., Galvin, P. B., & Gagne, G. (2014). 2.2.1 Command Interpreters. In Operating Systems Concepts. essay, Wiley.

[3] Yan, D. (2020, December 22). *Producer-consumer problem using mutex in C++*. Medium. Retrieved September 9, 2022, from https://levelup.gitconnected.com/producer-consumer-problem-using-mutex-in-c-764865c47483

[4] Yoo, S. H. (n.d.). Mobile-os-DKU-cis-MSE. GitHub. Retrieved September 8, 2022, from https://github.com/mobile-os-dku-cis-mse/