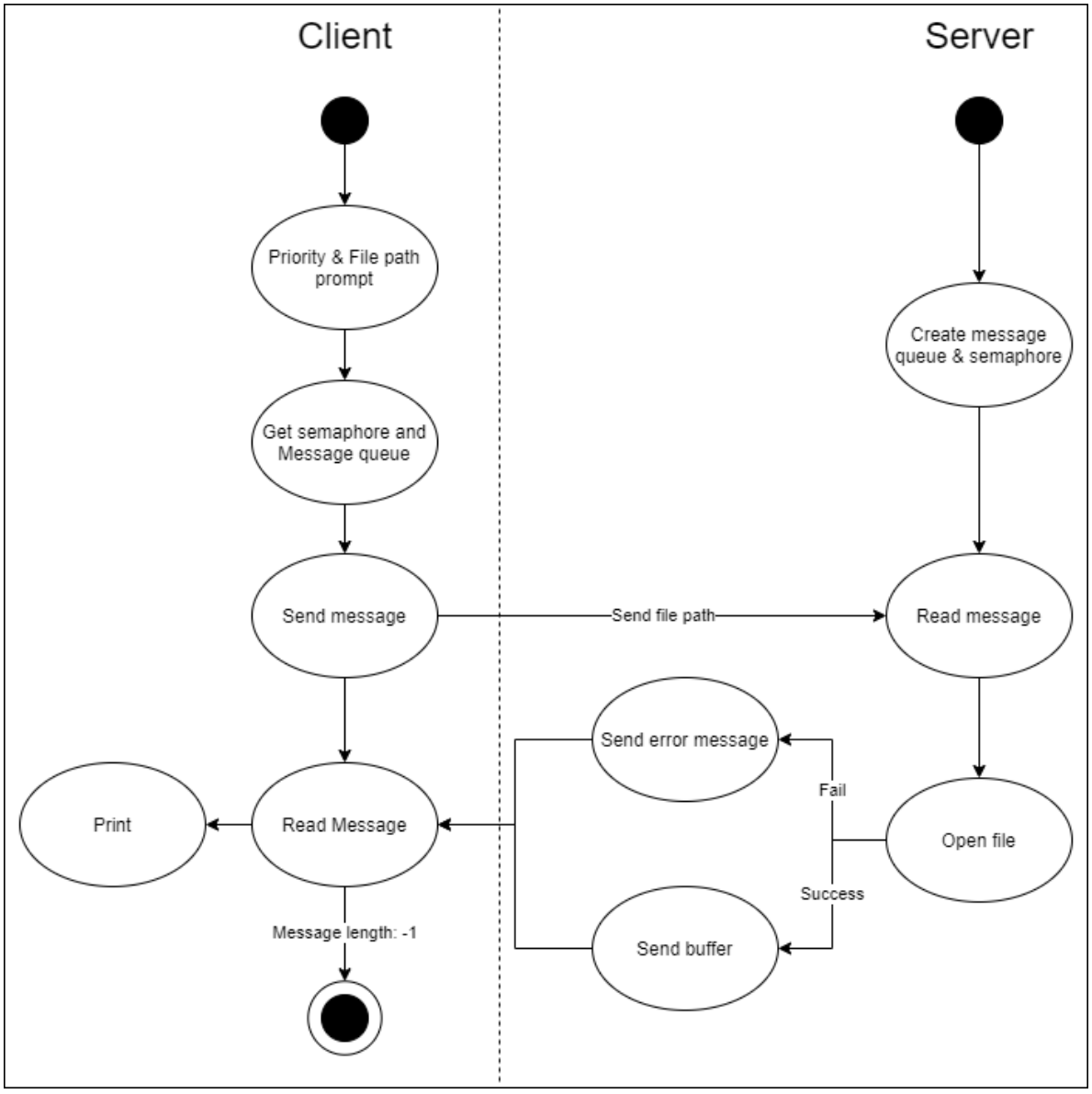
COMP4981 ASSIGNMENT2

INTERPOCCESS COMMUNICATION

SAM LEE

Table of Contents

Finite State Diagram



Pseudo Code

Server

Create message queue & semaphore

```
Create message queue{
        Create a message queue with predefined key
        If (a queue already exists)
                Get the id of the queue
}
Create semaphore{
        Create a semaphore with predefined key
        If (a semaphore already exists)
                Get the id of the semaphore
        Set a default semaphore value
}
```

Read message

```
Read message{
        While(true){
                Read message that has type 1
                If(message read)
                        Create a process to open the file and send the content back
        }
}
```

Open file

```
Open file{
        Open the file through the file path received
        Return whether it succeeded or failed to open the file
}
```

Send buffer

```
Buffer size change{
        Depending on the priority of the message, changes the buffer size.
        The lower the priority it has, the smaller buffer size.will be assigned
}
```

Send file{

While(end of file){

Read the file up to the specified buffer size

Wait for Semaphore

Send the buffer to client

Signal Semaphore

}

Send a message that has size -1 to indicate it's the end of file

}

Send error message

Send error message{

Send a message that has size -1 with error message

}

Client

Priority & File path prompt

Prompt priority and file path{

Get terminal argument

Set priority in message struct

Put file path into sending buffer

}

Get Semaphore and Message queue

Get semaphore{

Get a semaphore id with predefined key

}

Get message queue{

Get a message queue id with predefined key

}

Send message

Send message {

Wait semaphore

Send a message struct that is filled with file path, priority and type

Signal semaphore

Create a thread to print the content when a message is read

}

Read message

```
Read message{
        While(true){
                If('produced' flag is false){
                        Read message and store it into the reading buffer
                        If(message read)
                        Change 'produced' flag for printing thread
                }
        }
}
```

Design Decisions

Synchronization

In this program, shared resources that can be critical sections are two.

1. Message queue

2. Reading buffer in client

For the message queue, one semaphore is used to prevent multiple processes send messages at a time. Therefore, every time a process sends a message, that has to wait first and send it, then signals the semaphore. However, using one semaphore has a timing issue which is that a process wait and signal twice or more when another process does one time slice allocation on OS. However, it was noticed that it rarely happens, and it doesn't cause a noteworthy difference in the program's performance, so one semaphore was still used.

On the other hand, the reason why reading the queue is not synchronized is that reading the message queue doesn't interrupt other processes' reading. Every client tries to read a certain message that has process id as its message type, so there is no need for clients to take turns to read messages since it can only read its own message.

For the synchronization of reading buffer in client, it can be resolved by using two semaphores or using a flag in the shared structure. First of all, the reason why it is 'two' semaphores is it prevents the timing issues as described above, so it will look like as following,

Wait(s1)	Wait(s2)
Read()//fill the buffer	Print()
Signal(s2)	Signal(s1)

However, the flag method is chosen in this program because if we use two semaphore, every client will create two semaphores to communicate between the main process and thread. I considered it as a little overhead, so instead, I designed to create a client structure that has flag, so it lets the program read or print the buffer. It looks like as following,

If(produced == false)	If(produced == true)
Read()	Print()
produced == true	produced == false

However, it requires a while loop to check the ‘produced’ value, but I considered it’s a minor issues in this program reads the message fast enough and terminates when the all the text in a file is sent.