**CS3103 Project B Report**

**Group Information**

|  |  |  |  |
| --- | --- | --- | --- |
| Member | Name | SID | Email |
| #1 (contact person) |  |  |  |
| #2 | Mahir Labib | 56749556 | mlabib2-c@my.cityu.edu.hk |
| #3 |  |  |  |

Note: Please specify each team member's contribution if not all members make significant contributions to this project.

Table of Contents

[Problem 1 3](#_Toc164253604)

[2) Abstract idea and mechanism design. 3](#_Toc164253605)

[3) Implemented functions. 4](#_Toc164253606)

[3.1 schedulertest.c 4](#_Toc164253607)

[3.2 Makefile 4](#_Toc164253608)

[3.2 Syscall\_handler.c (Addition of New System Calls) 4](#_Toc164253609)

[**3.2.1 sys\_process\_prun** 4](#_Toc164253610)

[**3.2.2 Changes in existing functions** 5](#_Toc164253611)

[3.2.3 Prioritizing process creation in ‘sys\_process\_prun” 5](#_Toc164253612)

[3.2.4 Additional Error Checking 5](#_Toc164253613)

[3.3 runForSeconds() function 5](#_Toc164253614)

[3.4 list.c file 5](#_Toc164253615)

[**3.4.1 print\_list Function** 5](#_Toc164253616)

[3.4.2.1 List\_push\_head\_priority function 6](#_Toc164253617)

[3.4.2.1Change in Priority Ordering Logic 6](#_Toc164253618)

[3.5 Process.c file 6](#_Toc164253619)

[3.5.1 Change from **blocked\_list** to **ready\_list** for the priority launch 7](#_Toc164253620)

[**Before:** 7](#_Toc164253621)

[3.5.2 Process creation and Management 7](#_Toc164253622)

[1) Have you successfully implemented the named pipe into the basekernel? Have your test programs executed properly? If not, please provide potential reasons for the issues. 7](#_Toc164253623)

[2) Abstract idea and mechanism design. 8](#_Toc164253624)

[3 Implemented functions. 8](#_Toc164253625)

[3.1 named\_pipe.c file 9](#_Toc164253626)

[**3.1.1 ‘named\_pipe\_create(char \*fname)’** 9](#_Toc164253627)

[**3.1.2 ‘named\_pipe\_addref(struct named\_pipe \*p)’** 9](#_Toc164253628)

[**3.1.3 ‘named\_pipe\_flush(struct named\_pipe \*p)** 9](#_Toc164253629)

[**3.1.4 named\_pipe\_delete(struct named\_pipe \*p)** 9](#_Toc164253630)

[**3.1.6 ‘named\_pipe\_write\_nonblock(struct named\_pipe \*p, char \*buffer, int size)’** 10](#_Toc164253631)

[**3.1.7 named\_pipe\_read(struct named\_pipe \*p, char \*buffer, int size)** 10](#_Toc164253632)

[**3.1.8 ‘named\_pipe\_read\_nonblock(struct named\_pipe \*p, char \*buffer, int size)’** 10](#_Toc164253633)

[**3.1.9 ‘named\_pipe\_size(struct named\_pipe \*p)’** 11](#_Toc164253634)

# Problem 1

# 1) Have you successfully implemented the priority scheduling into the basekernel? Have your test programs executed properly? If not, please provide potential reasons for the issues.

Yes, we have implemented the priority scheduling into the base kernel and have properly tested out the programs and confirmed that they are executing properly.

# 2) Abstract idea and mechanism design.

# 3) Implemented functions.

## 3.1 schedulertest.c

**Schedulertest.c:** This code **automates** the creation of processes from specified executable files, assigning each a priority. It **opens each file**, **tries to run it as a process with the given priority**, and **reports success or error**. After setting up these processes, it initiates their execution.

* 1. **Purpose:** First, it prints the name of the process & forks a new process using **‘syscall\_process\_fork’.**
     1. **If it is the child process,** it runs the **function runForSeconds(int seconds**) is implemented with a different time for each process, i.e. ‘process5’ waits for 4 seconds, ‘process4’ waits for 1 second and so on.
     2. **If it is in the parent process,** it prints the child’s PID and waits for the child process to finish with the **“syscall\_process\_wait”**

## 3.2 Makefile

In the Makefile, we have added additional executable targets to the “**USER\_PROGRAMS**” variables, as these are the new user programs that we have developed and added to the build process. and they are as follows. ***‘process1.exe’, ‘process2.exe’, ‘process1.exe’, ‘process2.exe’, ‘process3.exe’, ‘process4.exe’, ‘process5.exe’, ‘schedulertest.exe’, ‘named\_pipe\_test.exe’.***

## 3.2 Syscall\_handler.c (Addition of New System Calls)

### **3.2.1 sys\_process\_prun**

Added to handle process creation with a specific priority, extending the functionalities of process management with priority considerations.  
A computer screen with white text

Description automatically generated

**Input Example**:

fd: File descriptor of a loaded executable

argc: 3

argv: {"./program", "arg1", "arg2"}

priority: 5

**Output Example**: Returns the PID of the child process if successful, or an error code like -1 for KERROR\_INVALID\_OBJECT if the file descriptor is invalid.

### **3.2.2 Changes in existing functions**

Enhanced error handling in ‘sys\_process\_run”  
A computer code on a black background

Description automatically generated

### 3.2.3 Prioritizing process creation in ‘sys\_process\_prun”

A black background with white text

Description automatically generated

### 3.2.4 Additional Error Checking

A black screen with white text

Description automatically generated

## 3.3 runForSeconds() function

It is almost the same as it is provided in the question. The **runForSeconds** function waits for a specified number of seconds using a **do-while** loop. It repeatedly checks the system time until the elapsed time matches the input parameter **seconds**.

## 3.4 list.c file

### **3.4.1 print\_list Function**

We introduced a function to print the priorities of the nodes in the list.

**Input**

**list**: A pointer to a struct list which represents the linked list. This structure should contain a head pointer pointing to the first node of the list.

* **Output**

**None**: The function does not return any value. It only outputs to the console.

### 3.4.2.1 List\_push\_head\_priority function

Allows pushing a node to the head of the list with a specified priority. This adds functionality that was not present in the older snippet.  
A computer screen with text and symbols

Description automatically generated

### 3.4.2.1Change in Priority Ordering Logic

In the older snippet (list\_push\_priority function), the condition for inserting based on priority was **pri > n🡪 priority**. In the newer snippet, this has been corrected to pri < n-> priority, implying the list maintains a sort order from lowest to highest priority, which was mentioned to be done in the question.  
Before:   
A black background with white text

Description automatically generated  
After:   
A black background with white text

Description automatically generated

## 3.5 Process.c file

Here, to accomodatre with the change from FIFO scheduling to priority scheduling, we needed to change the implementations of some of these files.

### 3.5.1 Change from **blocked\_list** to **ready\_list** for the priority launch

Before:

A black background with white text

Description automatically generated

**After:**

A black background with white text

Description automatically generated

### 3.5.2 Process creation and Management

We introduced priority during the process creation  
A black screen with white text

Description automatically generated

**Problem 2**

# 1) Have you successfully implemented the named pipe into the basekernel? Have your test programs executed properly? If not, please provide potential reasons for the issues.

# 2) Abstract idea and mechanism design.

# 3 Implemented functions.

**In this part of the report, we have compiled the functions added, in terms of the file names.**

## 3.1 named\_pipe.c file

### **3.1.1 ‘named\_pipe\_create(char \*fname)’**

**Purpose:** Creates and initializes a new named pipe structure.

**Input:** fname - A pointer to the name of the named pipe.

**Output:** Returns a pointer to a newly created named\_pipe structure if successful; otherwise, returns NULL.

* **Example:**
  + Input: "mypipe"
  + Output: Pointer to initialized named\_pipe structure.

### **3.1.2 ‘named\_pipe\_addref(struct named\_pipe \*p)’**

**Purpose:** Increments the reference count of the named pipe, indicating another part of the program is using the pipe.

**Input:** p - A pointer to the named\_pipe structure.

**Output:** Returns the same pointer passed as input after incrementing its refcount.

* **Example:**
* Input: Pointer to a named\_pipe
* Output: Same pointer with increased reference count.

### **3.1.3 ‘named\_pipe\_flush(struct named\_pipe \*p)**

**Purpose:** Flushes the named pipe by setting its flushed status, typically to allow any waiting processes to continue.

**Input:** p - A pointer to the named\_pipe.

**Output:** None.

### **3.1.4 named\_pipe\_delete(struct named\_pipe \*p)**

**urpose:** Decreases the reference count of a named pipe and frees its resources if the count reaches zero.

**Input:** p - A pointer to the named\_pipe.

**Output:** None.

3.1.5 ‘named\_pipe\_write(struct named\_pipe \*p, char \*buffer, int size)**’**

**Purpose:** Writes data to the named pipe in a blocking manner.

**Input:**

* p - Pointer to the named\_pipe.
* buffer - Pointer to the data buffer to be written.
* size - Size of the data to write.

**Output:** Returns the number of bytes written.

**Example:**

**Input**: Pipe pointer, "Hello", 5

**Output**: 5 (if all bytes were written)

### **3.1.6 ‘named\_pipe\_write\_nonblock(struct named\_pipe \*p, char \*buffer, int size)’**

**Purpose:** Writes data to the named pipe in a non-blocking manner.

**Input:** Same as named\_pipe\_write

**Output:** Returns the number of bytes written or 0 if no space is available

### **3.1.7 named\_pipe\_read(struct named\_pipe \*p, char \*buffer, int size)**

**Purpose:** Reads data from the named pipe in a blocking manner.

**Input:**

p - Pointer to the named\_pipe.

buffer - Pointer to the buffer where the data will be stored.

size - Maximum number of bytes to read.

**Output:** Returns the number of bytes actually read.

**Example:**

Input: Pipe pointer, buffer, 10

Output: Number of bytes read (depends on available data)

### **3.1.8 ‘named\_pipe\_read\_nonblock(struct named\_pipe \*p, char \*buffer, int size)’**

**Purpose:** Reads data from the named pipe in a non-blocking manner.

* **Input:** Same as named\_pipe\_read.
* **Output:** Returns the number of bytes read or 0 if no data is available.

### **3.1.9 ‘named\_pipe\_size(struct named\_pipe \*p)’**

**Purpose:** Retrieves the size of the named pipe, which is set to the page size (PAGE\_SIZE).

* **Input:** p - Pointer to the named\_pipe.
* **Output:** Returns the size of the pipe (PAGE\_SIZE).