# RTX Operating System Report

Tyler Babaran

insert student number here
EMAIL@uwaterloo.ca

Kelly McBride

insert student number here
EMAIL@waterloo.ca

Peter Socha

20484453
psocha@uwaterloo.ca

March 27, 2015

# Contents

# List of Algorithms

# List of Figures

# Chapter 1

# Introduction

Kelly

# Chapter 2

# Design Description

## 2.1 Global Variables and Structures

Kelly

## 2.2 Memory Management

### 2.2.1 Memory Structure

Tyler

### 2.2.2 Requesting Memory Blocks

Tyler

### 2.2.3 Releasing Memory Blocks

Tyler

## 2.3 Processor Management

### 2.3.1 Process Control Structures

Kelly

### 2.3.2 Process Queues

Tyler

### 2.3.3 Process Scheduling

Kelly

## 2.4 Process Priority Management

### 2.4.1 Get Process Priority

Peter

### 2.4.2 Set Process Priority

Tyler

## 2.5 Interprocess Communication

### 2.5.1 Message Structure

Kelly

### 2.5.2 Sending Messages

Tyler

### 2.5.3 Receiving Messages

Tyler

### 2.5.4 Delayed Send

Peter

## 2.6 Interrupts and I-Processes

### 2.6.1 UART I-Process

Peter

### 2.6.2 Timer I-Process

Peter

## 2.7 System Processes

### 2.7.1 Null Process

Peter

The null process has the lowest priority of any process in the operating system. It runs only when there are no ready processes to be run. When it runs, all it does is invoke `k_release_processor()` so that the kernel can check if there is a ready process to be run.

**Algorithm 1** The null system process

```
1: procedure NULL_PROCESS
2:     while true do
3:         K_RELEASE_PROCESSOR()
4:     end while
5: end procedure
```

### 2.7.2 KCD Process

Peter

The Keyboard Command Decoder process exists so that users can send console commands to the system at runtime. A command can be registered by sending the KCD process a `KCD_REG` type message. The KCD maintains a list of registered commands inside an array. When a `DEFAULT` command is sent to the KCD, the KCD will try to identify the message type, and will send the message to the appropriate process if it recognizes the command in its array. The KCD process is an intermediary between the UART i-process (which registers the keystrokes) and the eventual receiving message (which executes the command).

**Algorithm 2** The null system process

```
1: procedure NULL_PROCESS
2:     while true do
3:         K_RELEASE_PROCESSOR()
4:     end while
5: end procedure
```

### 2.7.3 CRT Process

Peter

The CRT process is used to print text to the system console. The process waits for messages of type `CRT_DISP`. If it receives such a message, it will send it to the UART i-process and modify the `IER` register so that the UART treats the message as an output message rather than an input. The UART will then output the text message to the console.

**Algorithm 3** The CRT Process

```
1:  procedure CRT_PROCESS
2:      while true do
3:          message = RECEIVE_MESSAGE()
4:          if message is of type CRT_DISPLAY then
5:              Send message to UART iprocess
6:              Set interrupt bits
7:          else
8:              RELEASE_MEMORY_BLOCK(message)
9:          end if
10:     end while
11: end procedure
```

## 2.8 User Processes

### 2.8.1 Wall Clock Process

**Peter**

### 2.8.2 Set Priority Process

**Peter**

### 2.8.3 Stress Test Processes

**Peter**

## 2.9 Initialization

**Kelly**

## 2.10 Testing

**Tyler**

```
/* PUT C Code here */
```

**Algorithm 4** This is an algorithm

```
1: function FOO
2:     for i = 0 to n do
3:         if i is prime then
4:             return i
5:         end if
6:     end for
7:     return not found
8: end function
```

# Chapter 3

# Major Design Changes

Add more sections as appropriate

## 3.1 Structure of Process Queue

Tyler

# Chapter 4

# Lessons Learned

**Everyone contribute something**

# Chapter 5

# Timing and Analysis

**This still needs to be programmed**