DESK No. | | | |

FAMILY NAME: ______________________

GIVEN NAMES: ______________________

SIGNATURE: ______________________

STUDENT NUMBER: | | | | | | | | |

THE UNIVERSITY OF WESTERN AUSTRALIA

SEMESTER 2, 2018 EXAMINATIONS

CITS2002

Physics, Mathematics & Computing

ECM

Systems Programming

Programming and Systems

This paper contains: **6** Pages **(including title page)**

Time Allowed: **2:00** hours

INSTRUCTIONS:
Answer all 6 questions. Each question is work 10 marks.

THIS IS A CLOSED BOOK EXAMINATION

SUPPLIED STATIONERY	**ALLOWABLE ITEMS**
1 x Answer Booklet 18 Pages	**UWA Approved Calculator with Sticker**

This page has been left intentionally blank

1) Consider the function `subString()`, whose C99 prototype follows:

```
bool subString(char *haystack, char *needle);
```

The purpose of the function is to determine if the character string pointed to by `needle` is a substring of the character string pointed to by `haystack`.

The function returns `true` if `needle` is a substring of `haystack` (i.e. `needle` completely appears within `haystack`), and `false` otherwise. You may assume that both `haystack` and `needle` are valid pointers to strings.

Write the function `subString()` in C99.

Do not simply call the similar C99 function `strstr()`.

(10)

---

2) Consider a 2-dimensional spreadsheet – a rectangular grid of cells, each of which holds an arbitrary string of characters, or an empty string if the value of the cell has not yet been defined. Each cell is identified by 2 non-negative integer values, identifying a specific row and column.

Define a C99 user-defined data type to represent an instance of the 2-dimensional spreadsheet.

Next, define and implement three C99 functions:

- one to return a pointer to allocated memory suitable for storing and managing the spreadsheet,
- one to swap the contents of two indicated columns in a spreadsheet, and
- one to completely deallocate all memory allocated to a spreadsheet.

(10)

SEE OVER

3) Consider environment variables used in Unix-based operating systems. A frequently seen environment variable is named `PATH`, and is used by command interpreters, or shells, to identify the names of directories to be searched to find executable programs.

For example, a typical value of `PATH` may be:

```
"/Users/chris/bin:/usr/local/bin:/usr/bin:/bin:."
```

which provides a colon-separated list of directories to search for a required program.

Write a C99 function, named `executeUsingPATH()`, which accepts the name of a program to execute and a NULL-pointer terminated vector of arguments to be passed to that program.

The requested program may be specified using just its name, or with either an absolute or a relative pathname.

```
int executeUsingPATH(char *programName, char *arguments[]);
```

Function `executeUsingPATH()` should attempt to execute `programName` from each directory provided via `PATH`, in order.

If `programName` is found and may be executed (passing to it the indicated program arguments), the function should wait for its execution to terminate, and then return the exit status of the terminated process. If the function cannot find and execute `programName` then it should simply return the integer -1.

Your function should not simply call the similar library function named `execvp()`.

(10)

4a) Explain clearly the following state transitions for processes and the reasons for the transitions:

1. from Running to Blocked.

2. from Blocked to Blocked-Suspend.

(5)

4b) Explain clearly the difference between a *program* and a *process*. What are the important pieces of information that a multi-processing operating system needs to save during process switching?

(5)

5a) Explain the importance of logical to physical address translation.

Consider a 16-bit computer in which the logical address has a 10-bit offset. Explain clearly, with the aid of a diagram, how the logical to physical address translation is performed for this computer.

What is the maximum number of pages that a process can be allocated in this computer?

(5)

5b) What are *system calls*, and why are they important in the design and implementation of an operating system?

Explain why almost every process needs to execute at least one system call during its execution.

(5)

---

6) Consider the following operating system dependent system-calls:

```
int unlink(char *filename);
int  rmdir(char *directoryname);
```

The `unlink()` system-call may be used to remove a file from its parent directory, and the `rmdir()` system-call may be used to remove a directory from its parent directory provided that `directoryname` itself contains no other files or directories.

Assume that both system-calls return 0 on success, and 1 on failure.

Using the `unlink()` and `rmdir()` system-calls, write a C99 function named `removeDirectory()` that removes the indicated (potentially non-empty) directory and all of its contents. Your function should have the following prototype:

```
int removeDirectory(char *directoryname);
```

You should assume `directoryname` contains only files and directories.

Your function should attempt to remove as many files and sub-directories as possible, returning 0 on complete success and non-zero otherwise. If `directoryname` could not be opened as a directory, your function should return -1.

(10)

END OF PAPER