Software Tools and Systems Programming — Midterm Test

Midterm Test

February 28th, 2024 – Duration: 2 hours

Total: 18 points

---

4 Problems

4 Questions

NAME	
Student Nbr	
Lecture Session	

Instructions

- Do **not** turn this page until you have received the signal to start.
- Please fill out the identification above and read these instructions.
- No aids are allowed.
- Please do not dettach any pages.
- Do not write on the QR code at the top of the pages.
- Lecture session is either: "Morning" (02) or "Afternoon" (01).

G o o d L u c k ! ! !

Problems

Problem #1 [4 pts.]

Given the dependencies for a given project as shown below, create a **makefile** consisting of rules capable of compile each single module, generate the executable and include as many other rules as *best practices* indicate. This is a scientific project which would certainly benefit from optimized calculations, as well as, clear indications of any suspicious compilation related issues.

`mySolver.c`	main driver program, depends on `ODEsolver.c`, `simulator.c` and `IOroutines.c`
`ODEsolver.c`	implements finite difference solutions, using Intel's Math Kernel Library (MKL)
`simulator.c`	implements a discrete time evolution marching scheme, using functions from the external libraries GSL and GSLcBLAS; and `ODEsolver.c`
`IOroutines.c`	Input/Output routines to read and write results using NETCDF format, provided by functions from the NETCDF library

You may assume that:

CC = gcc

CFLAGS = -Werror

- the GSL, GSLCBLAS, Intel's MKL and NETCDF libraries are installed and available on `/usr/local/lib`
- corresponding header files are available on `/usr/local/include`
- the corresponding names of the libraries are: `gsl`, `gslcblas`, `mkl`, `netcdf` respectively

Problem #2 [4 pts]

Write a C program that takes on the functionalities of the `wc` (word count) tool. I.e. the program will output the total number of characters, words and lines; depending on some command line arguments for a specified file or the standard input.

- If the optional argument `-c` is indicated then the program will display the total numbers of characters.
- If the optional argument `-w` is indicated then the program will display the total numbers of words.

  Words are separated by spaces. We will assume that not more than one space is present continuously.
- If the optional argument `-l` is indicated then the program will display the total numbers of lines.
- If not flagged argument is indicated, then the program will print the total number of characters, words and lines.

The program should also accept one file name or if not filename is indicated then it will take the input from standard input.

In case of any errors, these should be properly sent to the standard error channel.

```
wc [-c] [-w] [-l] [filename]
```

Problem #3 [3 pts]

Consider now that we are thinking of implementing our `wc` program from Problem #2, so that it can be used with substantially large files, i.e. that it has to be optimized to make the most efficient use of the computational resources employed in the code. What type of modifications would you suggest to implement so that the program runs more efficiently? In particular, consider the case of having to compute the 3 quantities (number of characters, words and lines). Provide a pseudo-code detailing your implementation, as well as, a detailed explanation of why this would be more efficient and how the implementation would work. The pseudo-code should have enough details so that it is clear what is being proposed and how it would be implemented.

Problem #4 [3 pts]

(a) Write the output of the following program.

(b) Draw a diagram of the processes that are created and identify which one is printing what.

(c) Explain if any of these processes can reach an "special" state. If that is the case, then explain how this can happen and what this/these state(s) is/are.

Assume that all processes run until they terminate normally.

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
  int p1, p2;
  printf(" A \n",);
  p1 = fork();
  if (p1 == 0) {
    printf(" B \n");
    p2 = fork();
    if (p2 == 0) {
      sleep(2);
      printf(" C \n");
      exit(0);
    }
    wait(0);
  }
  printf(" D \n");
  exit(0);
}
```

Output

Cont'd...

Questions

Getting full credit for the questions requires that you provide thorough explanations and detailed answers.

Question #1 [1 pt]

(a) What is an *inode*? (0.5 pts)

(b) Provide two examples of how this is used in Linux/Unix (0.3 pts)

(c) How would soft and hard links will be represented with inodes? (0.2 pts)

Question #2 [1 pt]

(a) Describe and provide details about the two main mechanisms for managing files in C/Unix (0.5 pts)

(b) What are "Standard File Descriptors"? Provide details and examples (0.5 pts)

Question #3 [1 pt]

Describe the *complete memory address model*, as discussed in class.

Provide details about the different components of the memory model, as well as, examples of variables declarations, memory allocations, *etc.* that show how these different components in the memory model are used.

Question #4 [1 pt]

(a) What is usually refer to as (raw) binary and text/ASCII *I/O formats*?

(b) Compare advantages and disadvantages of such representations.

(c) Suppose we have defined a new type of `semi-long int`, which uses 6 bytes. What range of numerical values for a variable of this `semi-long int` type uses: i) less, ii) equal, iii) more; bytes to store in binary format than in text format?
Justify your answer.

Scratch/Extra space

Cheat Sheet

Some Useful C Functions

int execl(const char *path, const char *arg0, ... /*, (char *)0 */);	
int execvp(const char *file, char *argv[])	
int fclose(FILE *stream)	
char *fgets(char *s, int n, FILE *stream)	
pid_t fork(void)	
FILE *fopen(const char *file, const char *mode)	
int fprintf(FILE * restrict stream, const char * restrict format, ...);	
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);	
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);	
pid_t getpid(void);	
pid_t getppid(void);	
int kill(int pid, int signo)	
void *malloc(size_t size);	
int open(const char *path, int oflag)	
int scanf(const char *restrict format, ...);	scans input according to a format
char *strncat(char *dest, const char *src, size_t n)	concatenate strings
char *strstr(const char s1, const char *s2);	locates the first occurrence of s2 in s1
int strcmp(const char *s1, const char *s2);	
int strncmp(const char *s1, const char *s2, size_t n)	compare strings
char *strncpy(char *dest, const char *src, size_t n)	copy strings
size_t strcspn(const char *s, const char *charset);	find offset of first character match
size_t strlen(const char *s)	string length
char *strpbrk(const char *s, const char *charset);	find characters in string
char *strrchr(const char *s, int c);	locate last occurrence of character in string
size_t strspn(const char *s, const char *charset);	find offset of first non-matching character
char *strtok(char *restrict str, const char *restrict sep);	string tokens
long strtol(const char *restrict str, char **restrict endptr, int base);	convert to long
int wait(int *status); int waitpid(int pid, int *stat, int options)	/* options = 0 or WNOHANG*/
ssize_t write(int d, const void *buf, size_t nbytes);	

Useful Macros

`WIFEXITED(status)` `WEXITSTATUS(status)` `WIFSIGNALED(status)` `WTERMSIG(status)` `WIFSTOPPED(status)` `WSTOPSIG(status)`

Make – Automatic variables

$@	The file name of the target of the rule.
$*	The file name of the target without the file extension.
$<	The name of the first prerequisite.
$?	The names of all the prerequisites that are newer than the target, with spaces between them.
$^	The names of all the prerequisites, with spaces between them. Discard duplicates.
$+	Similar to $^, but includes duplicates.