Write Up

We include <sys/wait.h>(wait system call is declared here) just so. However, since our system is POSIX compliant, it is not necessary to do so and the wait system call is defined as part of the kernel. In UNIX like systems, system calls are rather under the header of <unistd.h>.

Chosen\_process is the index of the currently chosen process from the array of processes

Fundamental flaw that I had which learnt of: You could pass a dynamically allocated array as Process arrP[] to a function. Took me a great deal.

int main(): For interactive input (e.g., scanf()).

int main(int argc, char \*argv[]): For command-line arguments input when running the program.

(command line input is provide in the command line or terminal while interactive input is given in response to the prompts within the program)

Question 1

We seed the random number generator, giving getpid and time(tells the number of seconds since Epoch) as arguments, as their sum would ensure randomness in output in each run of the program.

We note that the question asks us to execute wait 7 times so we need to make the child processes execute parallelly(and not in a serialized manner. So, we add an additional for loop towards the end instead of an else block in the second last loop(which would have had a wait(NULL) call).

Question 2

There would naturally be 4 forking levels as at the 4th level, we would be dealing with 1 element in each child process and we do not need to go beyond this. In case we do, then the first if statement of the binarySearch function(low > high) is there to see into the same. Since we want serialized execution of child processes, we have wait in an else block.

Question 3

*date.c*

time\_t data type, time and local time functions, strftime function,

We use fprintf instead of perror in the else clause, as the former is used in case of custom error messages while the latter in case of system related error messages that set the global variable: errorno. The file descriptor, stderr, by default directs to the console. argc[0] is always the program name.

Conversion characters:

a : abbreviated week day name(e.g., “Mon”)

b : abbreviated month name(e.g., “Jan”)

d : AM/PM

z : time zone offset from UTC(Coordinated Universal Time) in the format of +-HHMM(e.g., -0500 for UTC +5 hours and -0200 for UTC -2 hours)

Z : Time Zone

*cal.c*

Zeller’s congruence calculates day of week for the 1st of a particular month and year.

A black background with white letters

Description automatically generatedZeller's Congruence calculates the day of the week for a given date. It uses the formula:

**h**: Day of the week (0 = Saturday, 1 = Sunday, ..., 6 = Friday).

**q**: Day of the month.

**m**: Month (3 = March, 4 = April, ..., 14 = February).

**K**: Year of the century (year % 100).

**J**: Zero-based century (year / 100).

Adjustments are made for January and February by treating them as months 13 and 14 of the previous year.

We find first day of the month using Zeller’s congruence. The we print the calendar using the displayCalendar function.

atoi is under the stdlib.h header and has the signature: int atoi(const char \*str);. It returns 0 if the string cannot be converted to an integer.

*uptime.c*

sysinfo is a data structure that provides us information(statisitics) of the current state of the system. The uptime attribute of this tells of the time(in seconds) since the system was last booted. To use an instance of the sysinfo struct we use the sysinfo() method which fills an instance of the struct with the current system information. We pass the instantiated struct by reference. unsigned int is one of its multiple fields of sysinfo struct.

*main.c*

We use execlp from the exec family of 6 functions to search for the executable in the directories listed in the path environment variable. The first argument(to execlp) is the name of the program to be executed while the second one is the first argument passed to the program which we will now begin executing. Since execlp takes a variable number of arguments, we pass a null pointer((char\*)NULL) (a null pointer of type char\*)as the last argument to signify the end of arguments. In case of an error in execlp we print an error prefixed with “execlp” to indicate where the error occurred.

*Makefile*

We define the compiler, flags and targets. With each target we propose a rule to follow for its compilation. We then build executables for all the programs and add dependences to the executable of main.c. rm stands for remove and the flag “f” stands for force implying remove the mentioned files

and in case one does not exist, ignore it(do not produce an error). “f” also ensures that if any of these files are write-protected, they will be removed without asking for confirmation. rm -f should be used with caution as it removes the files from your system without any prompts. Say our source file is q1.c and its executable is q1 the running the command as in your code will remove the executable – this command works based on file name and not file type.

Question 4

--- Assuming time is taken in integer values as input ---

We choose process to execute based on the requirements of the algorithm.

**FIFO**

We find the process with the minimum arrival time using the findMinArrivalTimeProcess function. In case a process arrives such that the CPU had gone idle before that, we call the cupIdle function(Edge case) to see so. We update the process’s properties and have the desired output.

**SJF**

The outer while loop sees if all processes have executed. The inner for loop finds the one with least burst time. If no process is chosen we handle so(Edge case). We proceed to update process properties and have the desired output.

**SRTF**

We proceed a single unit of time at once to ensure correctness. Since we are now dealing with preemptive scheduling we have to cater to response time in a different manner. We update response time only when dealing with a process the first time.

**RR**

We execute processes as they come for a fixed time quantum. If the execution completes withing that, we leave that process and move to the next else we pause that process after the time quantum and move to cater to other processes. The inner for loop helps us model the ready queue; the outer while loop ensure all process are executed. Towards the end if no process got executed at a time instant, we proceed to the immediately next time instant(Edge case handling).