Part C:

FCFS

- The program starts in the main function, and I have declared two arrays, commands and arguments, are defined to hold the names of commands and their arguments and the variable num_commands is used to which commands to execute.
- for loop is used to iterate through the commands and arguments
  - Cpid variable is declared to store the child process ID, and a structure pstat_info is created to store process statistics.
  - child process is created using the fork system call. If the fork fails, an error message is printed in the terminal, and the program exits.
  - In the child process (when cpid == 0), the specified command is executed with its arguments using exec. If the execution fails, an error message is printed, and the child process exits.
  - In the parent process (when cpid > 0), the procstat function is called to get the process statistics for the child process.
  - The creation time, end time, and total time, wait time, turnaround time, average turnaround time, average wait time are printed on the terminal.
- The loop continues until all commands have been executed and their statistics have been displayed and the program exits.

Once the control is transferred to sysproc.c, the system call sys_procstat will execute.

- The sys_procstat function is a system call that is used to retrieve the process statistics and stores the statistics(createtime,endtime, totaltime, turn around time) in a pstat structure pointer.
- The variables pid and pstat to store the process ID and the pointer to the structure.
- We are using the argint and argptr functions to retrieve the process ID (pid) and the pointer to the pstat structure.
- If either of these retrieval operations fails (returns a negative value), the function returns -1 to indicate an error.
- It returns the result of the procstat function. If procstat succeeds, it returns the process ID; otherwise, it returns -1 to indicate an error.

Scheduler

- struct proc *minP = 0, *p = 0;: This line declares two pointers to struct proc named minP and p and initializes them to 0 (null).
- The code then enters a loop that iterates over the process table, looking for a process to run. It appears to be iterating through the ptable.proc array, which presumably contains information about all processes in the system.
- Within the loop, it checks if the current process p is in a RUNNABLE state. If not, it continues to the next process.

- It checks if minP is not equal to 0 (non-null). If it is not null, it compares the creation time (ctime) of the current process (p) with the creation time of minP. It's looking for the process with the lowest creation time, which essentially means the process that was created first.
- If minP is null (i.e., it's the first iteration of the loop), it sets minP to the current process p.
- After the loop, if minP is not null (i.e., a runnable process was found), it proceeds to switch to the chosen process:
  - It sets the current CPU's proc pointer to minP.
  - Calls switchuvm(minP) to switch to the virtual memory space of the selected process.
  - Changes the state of minP to RUNNING.
  - Performs a context switch using swtch(&(c->scheduler), minP->context). This effectively transfers control to the selected process's context, allowing it to run.

Here again the program control is transferred from sys_procstat to procstat function present in the proc.c and the create time, end time and total time fields are returned back to test.c in user program and the turnaround time an waiting time are calculated in test program

- acquiring a lock on the process table and function iterates through all processes in the process table.
- For each process, it checks if it's a child of the current process and matches the specified process id in the state.
- If there is a match found, with the help of ctime, etime, it calculates total time (ttime), if not it kills the current process.
- It releases resources for the process, sets its state to UNUSED, and returns the process id.
- If none of the above conditions are met, it puts the current process to sleep until a child exits, releasing the lock.

Priority Based Scheduling

- The program starts in the main function, and I have declared two arrays, commands and arguments, are defined to hold the names of commands and their arguments and the variable num_commands is used to which commands to execute.
- for loop is used to iterate through the commands and arguments
    - Cpid variable is declared to store the child process ID, and a structure pstat_info is created to store process statistics.
    - child process is created using the fork system call and also the priority is set using custmpro syscall(as the priority is pass in the array). If the fork fails, an error message is printed in the terminal, and the program exits.
    - In the child process (when cpid == 0), the specified command is executed with its arguments using exec. If the execution fails, an error message is printed, and the child process exits.
    - In the parent process (when cpid > 0), the procstat function is called to get the process statistics for the child process.
    - The creation time, end time, and total time, wait time, priority process id, turnaround time, average turnaround time, average wait time are printed on the terminal.
- The loop continues until all commands have been executed and their statistics have been displayed and the program exits.

Once the control is transferred to sysproc.c, the system call sys_procstat will execute.

- The sys_procstat function is a system call that is used to retrieve the process statistics and stores the statistics(createtime,endtime, totaltime, tatime) in a pstat structure pointer.
- The variables pid and pstat to store the process ID and the pointer to the structure.
- We are uses the argint and argptr functions to retrieve the process ID (pid) and the pointer to the pstat structure.
- If either of these retrieval operations fails (returns a negative value), the function returns -1 to indicate an error.
- It returns the result of the procstat function. If procstat succeeds, it returns the process ID; otherwise, it returns -1 to indicate an error.

Scheduler

- Initialization the variables: Declare two pointers, p and p1, for iterating through the process table. Obtain a reference to the current CPU and initialize its proc pointer to 0.
- Enter an infinite loop to continuously select processes to run and also Enable Interrupts:
- Allow the processor to receive hardware interrupts and context switches by enabling interrupts using sti().
- Here we are selection the priority Process Selection:
    - Initialize a pointer, highP, to represent the process with the highest priority.
    - Acquire the process table lock to ensure exclusive access to the process table.

    - Iterate through the process table to find a runnable process with the highest priority.
    - If a process is not in the RUNNABLE state, skip it.
    - Compare the priority of the current process (p) with the priority of other runnable processes (p1) to find the process with the highest priority.
- If a process with the highest priority (highP) is found, set p to this process, indicating it should be executed.
- Set the CPU's proc pointer to the selected process (p). Switch to the virtual memory space of the selected process using switchuvm(). Change the state of the selected process to RUNNING. Use swtch() to perform the context switch to the selected process's context, allowing it to run.

Here again the program control is transferred from sys_procstat to procstat function present in the proc.c and the create time, end time and total time fields priority process pid, wait time, turnaround time are returned back to test.c in user program and the turnaround time an waiting time are calculated in test program

- acquiring a lock on the process table and function iterates through all processes in the process table.
- For each process, it checks if it's a child of the current process and matches the specified processid in the state.
- If there is a match found, with the help of ctime, etime, it calculates total time (ttime), if not it kills the current process.
- It releases resources for the process, sets its state to UNUSED, and returns the processid.
- If none of the above conditions are met, it puts the current process to sleep until a child exits, releasing the lock.