Main Function

Description:
The primary entry point of the program. It initializes `NCPU` and `TSLICE` values using command-line arguments, sets up signal handlers, initializes the scheduler, and enters the infinite user input loop.

Parameters:

- `argc`: The number of command-line arguments.
- `argv`: Array of command-line arguments.

Returns:

- `1` if the number of arguments isn't 3.
- `0` after completion.

ExitTheShell

Description:
Gracefully exits the shell by printing an exit message, clearing the process queue, killing the scheduler process, and exiting the program.

Parameters:

- `sig`: The signal number, used for handling SIGINT (Interrupt signal, usually from a Ctrl+C).

ClearFile

Description:
Releases the memory occupied by the process and running queues.

---

read_user_input

Description:
Reads a line of input from the user using the `getline` function.

Returns:

- A pointer to the input string.

StringToArray

Description:
Converts an input string into an array of words by splitting the string based on spaces.

Parameters:

- `input`: The string to be split into words.

Returns:

- An array of words.

---

infinite_input

Description:
Prompts the user indefinitely for commands, handling special commands like "submit" and "run", and uses the `ExecuteCommands` function for other commands.

ExecuteCommands

Description:
Executes a provided command using a child process. The parent process waits until the child process finishes executing the command.

Parameters:

- `cdsArr`: An array containing the command and its arguments.

Process

Description:
This structure represents a process with its process id, command to execute, and a boolean flag indicating whether it's new or not.

Fields:

- `pid`: The process ID.
- `command`: A string containing the command that this process represents.
- `new`: A flag to indicate whether this is a new process or not.

QueueNode and Queue

Description:
The `QueueNode` structure represents a node in the linked list, whereas the `Queue` structure represents the entire linked list that functions as a queue.

Fields (QueueNode):

- `process`: A pointer to the `Process` structure.
- `next`: A pointer to the next node in the queue.

Fields (Queue):

- `front`: A pointer to the front of the queue.
- `rear`: A pointer to the rear of the queue.

Globals

Description:
Global variables that store values like the number of CPUs, time slice value, alarm time, scheduler's process ID, and pointers to the process and running queues.

Initialise

Description:
Initializes the `process_queue` and `running_queue`.

---

create_new_process

Description:
Allocates memory for a new process, sets its command and PID, and marks it as new.

Parameters:

- `pid`: The process ID.
- `command`: The command for the process.

Returns:

- A pointer to the newly created process.

---

enqueue

Description:
Adds a process to the end of a specified queue.

Parameters:

- `p`: The process to be added.
- `q`: The queue to which the process should be added.

---

dequeue

Description:
Removes a process from the front of a specified queue and returns it.

Parameters:

- `q`: The queue from which the process should be removed.

Returns:

- A pointer to the removed process node.

---

show_queue

Description:
Displays the current status of a specified queue.

Parameters:

- `q`: The queue to display.

---

destroyQueue

Description:
Frees up the memory occupied by a queue.

Parameters:

- `q`: The queue to be destroyed.

scheduler

Description:
Manages the processes by moving them between queues, starting new processes, stopping currently running processes, and ensuring that processes are scheduled based on the number of available CPUs and the time slice.

Parameters:

- `num_procs`: The number of processes to be managed by the scheduler.

---

handler_for_alarm

Description:
A signal handler function that gets called when an alarm goes off. Invokes the scheduler to manage processes.

● Contribution Percentage:
50% Details:
➔ Made a basic structure of the code, laying down the logic and components.
➔ Focused on the code implementation, from defining global variables to drifting the initial versions of the primary functions.

2. Arav: ● Contribution Percentage: 50% Details:
➔ Worked intensively on error analysis.
➔ Dedicated efforts towards the optimization of the code, making it efficient, and more user friendly.

Final Breakdown:
● Anish: 50%
● Arav: 50%

GITHUB : https://github.com/ianishdev/OS-Assignment.git