Files and their functionality:

1. Main.c:

* **Initialization and Iteration Setup:**
  1. Initialize variables, including the number of simulation iterations (simulationIterations), the number of scheduling algorithms (algorithms), and the time slice for the Round Robin algorithm (timeSlice).
  2. Create arrays of structures (Average\_Statistics) to store statistics for each scheduling algorithm (fcfs\_stats, sjf\_stats, srf\_stats, roundrobin\_stats, hpfp\_stats, hpfnp\_stats, final\_results).
* **Simulation Loop:**
  1. Iterate through the simulation for the specified number of iterations (simulationIterations).
  2. Generate a linked list of processes using the generate\_processes function and print details of the processes. -> "Process ID | Arrival Time | Run Time | Priority |”.
  3. Execute each scheduling algorithm for the current set of processes and store the resulting statistics in their respective arrays (fcfs\_stats, sjf\_stats, etc.).
* **Result Aggregation Loop:**
  1. Initialize an array (final\_results) to store the average statistics for each algorithm.
  2. Sum up the statistics from each iteration for all algorithms separately (response time, wait time, turnaround time, throughput).
* **Average Computation:**
  1. Calculate the average statistics for each scheduling algorithm by dividing the summed values by the number of simulation iterations.
* **Print Results:**
  1. Print the average statistics for each scheduling algorithm, organized in a clear format.

1. Process.h and process.c:

* **Header File Purpose:**
  1. The "Process.h" header file declares the structure struct process\_s, which represents a process with attributes such as process ID (pid), arrival time (arrival\_time), run time (run\_time), and priority. This header file provides the interface for creating, copying, and generating processes.
* **Function Declarations:**
  1. The header file declares three functions: create\_process, get\_copy\_of\_process, and generate\_processes. These functions are responsible for creating a process, generating a copy of a process, and generating a linked list of processes, respectively.
* **Linked List Sorting Function:**
  1. The header file includes a function pointer (compare) and uses it for sorting the processes in a linked list. The comparison function compares processes based on their arrival times, facilitating the sorting process in the linked list.
* **Linked List Generation in Source File:**
  1. The "process.c" source file defines the generate\_processes function, responsible for creating a specified number of processes with random attributes (arrival time, run time, and priority) and adding them to a linked list. The processes are then sorted based on their arrival times.

1. Utility.h and utility.c:

Utility.h:

1. Declares Node and Linked\_List structures.
2. Provides functions for linked list operations and queue management.

Utility.c:

* **Linked List and Node Creation:**

1. create\_Linked\_List initializes a new linked list structure.
2. create\_Node creates a node with the provided data.

* **Node Addition and Removal:**

1. add\_Node adds a node to the end of the linked list.
2. remove\_Node removes a specified node from the linked list.
3. remove\_head removes the head node from the linked list.

* **Sorting Linked List:**

1. sort function sorts the linked list using a comparison function.

* **Queue Implementation:**

1. create\_Queue creates a queue using the linked list structure.
2. enQueue adds an element to the end of the queue.
3. deQueue removes and returns the element from the front of the queue.
4. isEmpty checks if the queue is empty.

* **Process Statistics Printing:**

1. print\_policy\_stat prints detailed statistics for processes in a linked list.
2. Calculates and returns average statistics for response time, wait time, turnaround time, and throughput.
3. **stats.h:**

* **Structure Definitions:**
  1. Declares Process\_Statistics structure to store detailed statistics for a process.
  2. Defines Average\_Statistics structure for storing average metrics over multiple iterations.
* **Average Metric Declarations:**
  1. Declares average statistics functions for different scheduling algorithms, both preemptive and non-preemptive.
* **Algorithm-Specific Functions:**

1. Functions like first\_come\_first\_serve\_nonpreemptive, shortest\_job\_first\_nonpreemptive, etc., calculate and return average statistics for specific scheduling algorithms.

* **Priority Scheduling with Aging:**

1. Introduces priority scheduling with aging through functions highest\_priority\_first\_preemptive\_aging and highest\_priority\_first\_nonpreemptive\_aging.

* **Printing Function:**

1. Declares print\_policy\_stat function to print detailed statistics for processes in a linked list, including response time, wait time, turnaround time, and priority.
2. Algorithm definitions:

* **FCFS (First-Come-First-Serve):**
  1. FCFS is a non-preemptive scheduling algorithm where processes are executed in the order they arrive, with the first process in the ready queue being the first to execute.
* **SJF (Shortest Job First):**
  1. SJF is a non-preemptive scheduling algorithm that selects the process with the shortest burst time first, aiming to minimize the total processing time.
* **SRTF (Shortest Remaining Time First):**
  1. SRTF is a preemptive version of SJF, where the scheduler can preemptively switch to a process with a shorter remaining burst time if it becomes available.
* **Round Robin:**
  1. Round Robin is a preemptive scheduling algorithm where each process is assigned a fixed time slice or quantum, and they take turns executing in a circular order. If a process's time slice expires, it is moved to the back of the queue.
* **Priority non-preemptive:**
  1. Priority Scheduling is a non-preemptive algorithm where each process is assigned a priority, and the process with the highest priority is selected for execution first. If two processes have the same priority, FCFS order is used to break ties.
* **Priority Preemptive:**
  1. Priority Preemptive Scheduling is similar to Priority Non-Preemptive, but a higher-priority process can preempt the execution of a lower-priority one if it becomes available in the ready queue.

1. Brief definitions of metrics:

* **Response Time:**
  1. **Definition:** Response time is the time elapsed between submitting a request and receiving the first response. In a scheduling context, it's the time from process arrival to the start of its execution.
  2. **Avg Response Time:** Average Response Time is the mean of response times for all processes in a system. It indicates the average time taken for a process to start execution after its arrival.
* **Turnaround Time:**
  1. **Definition:** Turnaround time is the total time taken by a process to complete its execution, from arrival to termination.
  2. **Avg Turnaround Time:** Average Turnaround Time is the mean of turnaround times for all processes. It represents the average time a process spends in the system.
* **Wait Time:**
  1. **Definition:** Wait time is the total time a process spends waiting in the ready queue before it starts executing.
  2. **Avg Wait Time:** Average Wait Time is the mean of wait times for all processes. It indicates the average time a process spends waiting in the ready queue.
* **Throughput:**
  1. **Definition:** Throughput is the number of processes that complete their execution in a unit of time. It measures system efficiency.
  2. **Avg Throughput:** Average Throughput is the mean of throughput values over multiple runs or iterations. It provides a more stable representation of system performance.