Report : Priority Based Scheduler in xv-6

Functions in proc.c

1)int min(int a,int b) :
Compares two integers and returns the smaller of the two.

2)int _rbi(struct proc *p) :
Checks if the process's RBI is already set to 25. If so, returns 25 (default value) and resets the RBI.
Calculates RBI using a weighted sum involving running time, sleeping time, and waiting time.
Ensures that the calculated RBI is non-negative.

3)void scheduler(void):

Purpose:
The scheduler function serves as a preemptive priority-based scheduler, determining the process with the highest priority for execution.

Loop Structure:
Operates within an infinite loop, continually seeking the highest-priority process for scheduling.
Enables interrupts at the beginning of each iteration to prevent potential deadlocks.
Priority Calculation:

Utilizes a priority evaluation mechanism involving SP, DP, and other tie-breaking factors.
Maintains a reference to the highest-priority process (high_pri).
Process Selection:

Acquires the lock for each process to ensure data consistency.
Checks if a process is in the RUNNABLE state.
Calculates the current RBI (curr_rbi) for the process before making priority comparisons.

p->curr_rbi = _rbi(p); // find curr_rbi before comparing
Compares the priority of the current process with the current highest-priority process (high_pri).
Selects the process with higher priority based on SP, DP, number of times scheduled (tot_sche), and creation time (ctime).
Process State Update and Resource Allocation:

Updates the state of the selected process to RUNNING.
Allocates CPU resources to the selected process using swtch.
Updates the total scheduling time (tot_sche).
Releases the lock for the selected process.
Rescheduling:

Continues the loop to find the next process to execute.

Execution of high_pri for 1 Quantum Time Slice:

If a process is selected (high_pri is not null and is in the RUNNABLE state), the process is marked as RUNNING, and CPU resources are allocated for one quantum time slice using swtch.

System Call- set_priority() in sysproc.c

1. Purpose:
The implemented sys_set_priority function and the corresponding set_priority system call aim to allow users to change the static priority of a process and trigger rescheduling if necessary. The system call returns the old static priority of the process.

2. Function Signature:
uint64 sys_set_priority(void) system call
3. Implementation Details:
Inputs:

pid: Process ID of the target process.
new_priority: New static priority to be set.
Functionality:

Uses the argint function to retrieve input arguments from the user.
Validates that the new priority is within the acceptable range (0-100).
Iterates through the process table to find the target process pid.
Acquires the process lock, updates the static priority, resets the RBI to 25, and checks if rescheduling is required.
Releases the lock and, if needed, triggers a "yield" to allow the scheduler to reassess priorities.
Return Value:

Returns the old static priority of the process.
Returns -1 if the specified process ID does not exist.

b. set_priority System Call:
The system call invokes the sys_set_priority function using a software interrupt to transition to kernel mode.

Error Handling:
Checks for valid input parameters and returns appropriate error codes (-1) when necessary.

setpriority.c/user -
1. Purpose:
The purpose of the setpriority user program is to provide a convenient interface for users to change the priority of a process using the set_priority system call.

2. Program Structure:
Function Signature:
int main(int argc, char *argv[])
Inputs:
argc: Number of command-line arguments.
argv: Array of command-line arguments.

3. Command-Line Arguments Handling:
Ensures that the program is called with the correct number of arguments (2).

Parses the provided command-line arguments using atoi to obtain the process ID (pid) and the new priority value (priority).

4. Validation:
Verifies that the provided priority is within the acceptable range [0, 100].
Displays an error message and exits the program if the input parameters are invalid.

5. System Call Invocation:
Calls the set_priority system call, passing the parsed pid and priority as arguments.
Captures the returned old priority value.

6. Output Handling:
If the return value is -1, indicates that the process with the specified PID does not exist or the priority is invalid.
Otherwise, prints the old priority and the updated priority of the process.

7. Example Output:
Displays informative messages about the old and new priorities of the targeted process.

Conclusion:

RR: Average rtime 12, wtime 75
PBS: Average rtime 11, wtime 77

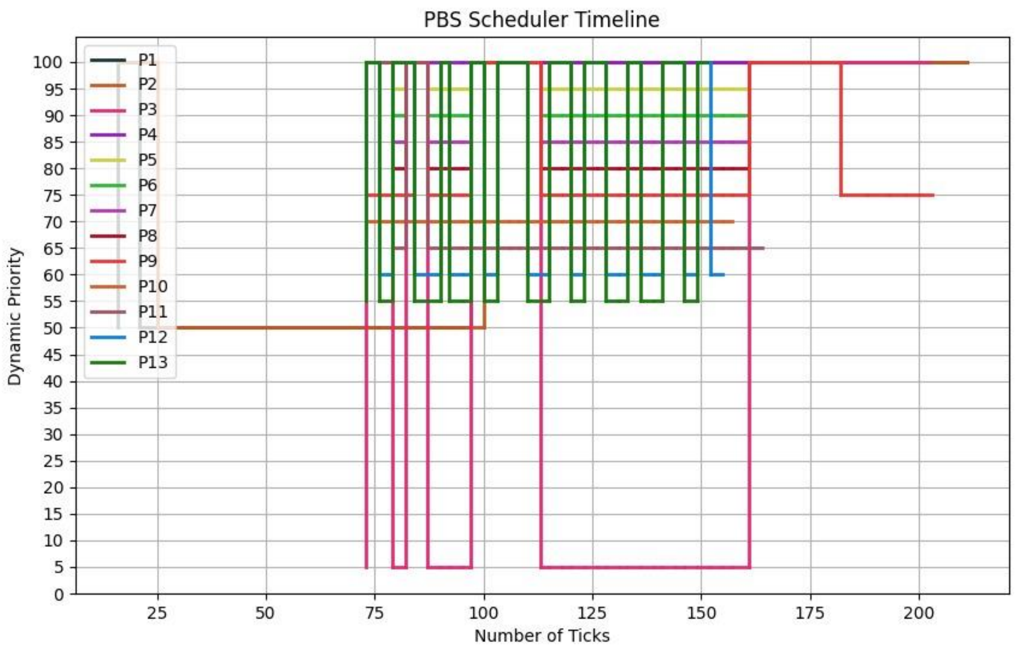
Static Priority (SP): Represents the inherent priority of a process, ranging from 0 to 100, where lower values indicate higher priority.
The default SP is set to 50.

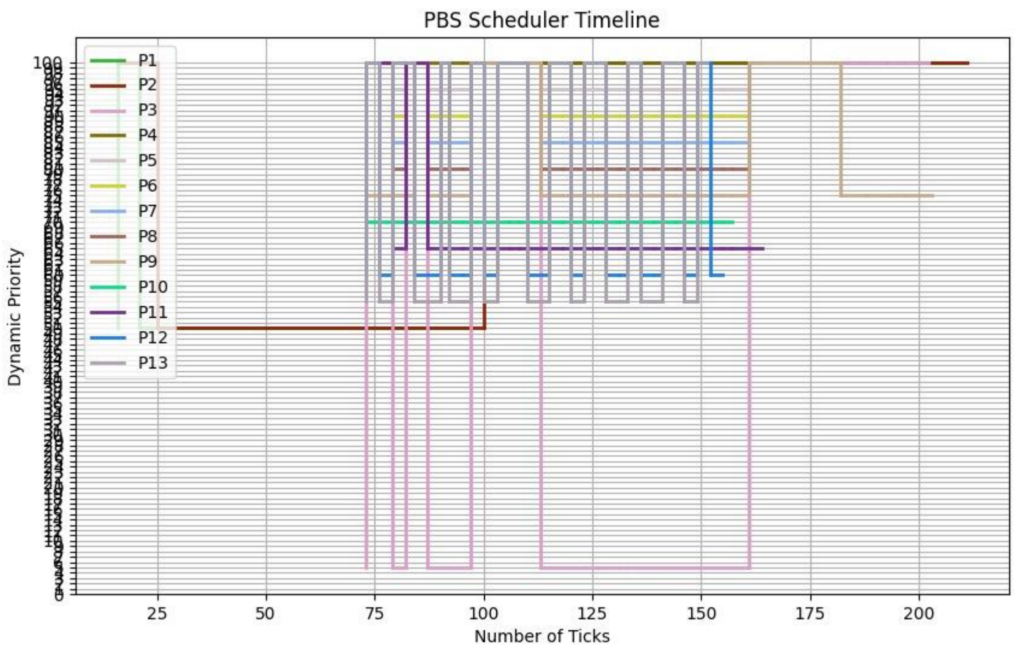
Recent Behaviour Index (RBI): Reflects the recent behaviour of a process, serving as a weighted sum of Running Time (RTiime), Sleeping Time (STime), and Waiting Time (WTime).
The default RBI is set to 25

Reasons for Using DP in Scheduling:

Adaptability: DP dynamically adjusts to changes in a process's behaviour, ensuring adaptability to evolving resource requirements.

Incorporation of SP: By incorporating SP into DP, the scheduler respects the inherent priorities of processes, offering a balanced approach that considers both short-term and long-term behaviour.


PBS Scheduler Timeline
Dynamic Priority
Number of Ticks
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13


PBS Scheduler Timeline
Dynamic Priority
Number of Ticks
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13