Università della Svizzera italiana

Faculty of Informatics

Operating Systems

PintOS: Report Project **3**

Group #1

Ivan Linnikov, Mariia Batalkina

Spring Semester, March 28, 2024

Report Instructions

- Please report **all** changes, even if minor, that you did to complete the project.
- You have to list all files that have been modified, and for each of them, list all functions/structs that have been modified or added (clearly stating "modified"/"added"). Then add a brief explanation or motivation for all those changes.
- A single report is required for each group. For the first individual project, each student submits a report together with the source code files that were changed.

1 Files changed

- /pintos0/pintos-env/pintos/threads/thread.c
- /pintos0/pintos-env/pintos/threads/thread.h

2 Changes

/pintos0/pintos-env/pintos/threads/thread.c

- next thread to run(void) (*modified*):

  In summary, I added code to temporarily disable interrupts before accessing and modifying the readylist. This action ensures the atomicity of these operations, making them interrupt-safe and preventing potential race conditions. Once the operations are completed, interrupts are re-enabled to allow the resumption of interrupt-driven tasks and preemption, maintaining the integrity of the scheduler's operations.

- thread get load avg(void)(*added*):

  In summary, the modification to the threadgetloadavg() function aligns with the task requirement by implementing the computation of the load average, as specified by the Advanced Scheduler specifications. By incorporating this functionality, the function contributes to the overall robustness and reliability of the scheduler's operations. Additionally, the conditional return statement ensures that the function behaves appropriately depending on the system configuration, thereby enhancing its reliability and adaptability to different runtime scenarios.

- thread get recent cpu(void) (*modified*):

  In summary, the modified thread get recent cpu(void) function fulfills the task requirement by implementing the computation of the recent CPU usage for the current thread, as specified by the Advanced Scheduler specifications. This implementation enhances the functionality of the scheduler by providing access to crucial information about thread performance. Additionally, the conditional return statement ensures that the function behaves appropriately based on the system configuration, thereby improving its reliability and adaptability to different runtime scenarios.

- thread set priority(int new priority) (*modified*):

  In summary, the modified thread set priority() function now includes additional logic to handle priority changes effectively, aligning with the requirements of the Priority Scheduler. By incorporating these enhancements, the function ensures that thread priority changes are appropriately managed, preventing lower-priority threads from monopolizing the CPU when higher-priority threads are ready to run. This implementation improves the fairness and responsiveness of the scheduler, enhancing overall system performance. Additionally, the conditional execution of the priority adjustment logic based on the thread mlfqs flag ensures compatibility with different scheduler configurations, thereby improving the function's versatility and robustness.

- thread create(const char *name, int priority, thread func *function, void *aux (*modified*):

  In summary, the modification to the thread create() function introduces additional logic to handle thread creation in accordance with the requirements of the Priority Scheduler. By including this enhancement, the function ensures that newly created threads with higher priorities preempt the current thread, thereby preventing potential priority inversion scenarios. This implementation improves the responsiveness and fairness of the scheduler, enhancing overall system performance. Additionally, the conditional execution of the thread yielding logic ensures that higher-priority threads have the opportunity to execute promptly, aligning with the principles of priority-based scheduling.

- thread set nice(int) (*modified*):

  In summary, the modification to the thread set nice() function introduces additional logic to handle changes to a thread's "nice" value in accordance with the requirements of the Advanced Scheduler. By incorporating this enhancement, the function ensures that the thread's priority is recalculated based on the new "nice" value, and if necessary, the thread yields the CPU to higher-priority threads. This implementation enhances the responsiveness and fairness of the scheduler, as threads with adjusted "nice" values can dynamically adapt to the system's workload. Additionally, the conditional execution of the thread yielding logic ensures that higher-priority threads have the opportunity to execute promptly, aligning with the principles of priority-based scheduling.

- thread get nice(void) (*modified*):

In summary, the modification to the thread get nice() function aligns with the task requirement by implementing the retrieval of a thread's "nice" value, as specified by the Advanced Scheduler specifications. By incorporating this enhancement, the function provides access to crucial information about a thread's scheduling priority, allowing for better system management and analysis. Additionally, the conditional return statement ensures that the function behaves appropriately based on the system configuration, thereby improving its reliability and adaptability to different runtime scenarios.