Department of Electrical and Computer Engineering
North South University (NSU)

CSE 323: Operating Systems Design
Section 10

Final Project Report

Instructor: Md Salman Shamil Semester: Fall 2025

Title:	Implementing Process Statistics System Call and Heap Growth Tracking System Call
Name:	Hasnat Karibul Islam
NSU ID:	2211275042
Name:	Qm Asif Tanjim
NSU ID:	2211402042
Name:	Nowren Mah Jabin Khan
NSU ID:	2211129642

Final Project Report

Title: Implementing Process Statistics System Call and Heap Growth Tracking System Call

Introduction:
Effective process management, memory utilization, and equitable resource scheduling are fundamental aspects of contemporary operating systems. The xv6 kernel, a lightweight, pedagogical Unix-like operating system, serves as an excellent environment for experimenting with kernel-level ideas and understanding core OS principles.

This project was undertaken to significantly enhance the observability and debugging capabilities of xv6, addressing its inherent lack of transparency regarding dynamic memory usage and global process state. To achieve this, we successfully developed and implemented two new kernel features: the Heap Growth Tracking System Call **(hapinfo)** and the Process Statistics System Call **(getpinfo)**.

The integration of these system calls transforms xv6 from a system that merely executes programs into a system that transparently reveals the inner workings of memory management and process life cycles, thereby improving support for debugging, experimentation, and learning.

Project Goals:
Our project focused on implementing two key features to extend the functionality if the xv6 kernel:

1. **Heap Growth Tracking System Call (hapinfo)**: This feature records every heap operation (expansion or shrinkage via **sbrk()**) performed by a process, along with its precise system timestamp **(ticks)**. This chronological log is then exposed to the user process for analysis.

2. **Process Statistics System Call (getpinfo):** This feature exposes detailed runtime statistics for all running processes (including PID, state, memory size, and potential scheduling metrics) through a structured data return.

The default xv6 operating system lacks the necessary infrastructure for these features:

- **Heap Tracking:** While **sbrk()** allows processes to manage their heap, the kernel only maintains the current process size. It has **no built-in log** or mechanism to track the history of heap adjustments.
- **Process Statistics**: xv6 processes can query basic information like their own PID, but the kernel lacks a single system call capable of querying the state, memory usage, and performance metrics of all other processes simultaneously.

The successful implementation of both system calls significantly improves the xv6 kernel:

- **Enhanced Debugging: hapinfo** provides crucial transparency into memory behavior, allowing developers to pinpoint the exact sequence and timing of heap operation.

- **Global Monitoring: getpinfo** enables user-space tools to monitor status and resources consumption of all active processes, essential for understanding system health and scheduling efficiency.
- The feature collectively make xv6 a more robust platform for developing and testing system-level applications.

Modifications:

What challenges arose when implementing the new features:

Our main challenge was safely embedding persistent logs and global data access into the core kernel structures while ensuring concurrency safety.

For **hapinfo**, we successfully overcame the following:

1. **Extending the Process Control Block (PCB):** We extended the fixed-size **struct proc** in **proc.h** to house the log array **(haplog[HAPLOG_SIZE])** and its index **(haplog_index).**

2. **Atomic Logging:** We ensured that reading and logging the system time (**ticks**) within **sys_sbrk()** was protected from race conditions by using **acquire(&tickslock)** and **release(&tickslock)**.

For getpinfo, the primary challenge was:

1. **Global Lock Management:** We implemented logic within **sys_getpinfo** to safely lock the process table, iterate through all process entries, gather their statistics, and release the lock quickly to minimize latency for other processes.

System calls and purposes each one serve:

We successfully implemented both system calls, requiring updates to **syscall.h**, **syscall.c**, and **usys.pl** for system call number assignment and stub creation:

System Call	System Call Number	Purpose
hapinfo(struct hapinfo *hi)	SYS_hapinfo	Copies the recorded log of heap operations from the current process's kernel space to the user-provided struct hapinfo buffer.
getpinfo(struct pstat *p)	SYS_getpinfo	Gathers statistics on all running processes and copies the data into the user-provided struct pstat buffer.

Modifications we made to the kernel:

Modifications for getpinfo (Process Statistics):

1. **pstat.h**: We created this file to define **struct pstat**, which contains an array of per-process statistical entries.

2. **kernel/proc.h:** We extended **struct proc** with new fields to maintain the necessary statistical data for each process.

3. **kernel/sysproc.c (sys_getpinfo)**: We implemented this function to iterate over the entire process table, gather the required statistics under the necessary lock, and use **copyout()** to transfer the global process data array to the user-provided buffer.

Modifications for hapinfo (Heap Tracking):

hapinfo.h: Defined **struct haplog** and **struct hapinfo**.

kernel/proc.h: Modified **struct proc** to include the **haplog** array and **haplog_index**.

kernel/sysproc.c (sys_sbrk): Injected logging logic to record events upon successful heap modification.

kernel/sysproc.c (sys_hapinfo): Implemented the new system call function to safely transfer the log data using **copyout()**.

Modifications:

We validated both implemented features by confirming the integrity and correct transfer of their respective data structures:

- **hapinfo validation**: We verified that **sys_sbrk()** accurately logged size changes and corresponding **ticks** values, and that **hapinfo()** safely transferred the complete, ordered log.

- **getpinfo Validation**: We confirmed that **sys_getpinfo()** correctly iterated through all processes, gathered their current statistics, and transferred the entire array of process statistics to user space without error.

User programs we wrote to test the new feature:
To test the new features have developed two user level test programs:

- **hapinfo_test.c**: Used to exercise **sbrk()** multiple times and then call **hapinfo()** to retrieve and print the heap operation log.

- **pstat_test.c**: Used to spawn several background processes, call **getpinfo()**, and display a comprehensive table of all running processes and their collected statistics.

Outcomes from these user programs demonstrate the new kernel capabilities:

- **Heap Tracking Proof**: **hapinfo_test** successfully printed a clear, chronological table of heap operations, confirming the kernel's enhanced memory observability.

- **Process Statistics Proof**: **pstat_test** successfully displayed a formatted table listing all active processes, their PIDs, their current memory usage, and their state, confirming the functional integrity of the getpinfo system call.

Conclusion:
We have successfully implemented both system calls as proposed. However, our focus shifted toward robust functionality over performance optimization. While the **getpinfo** system call is functional, its primary area for improvement relates to concurrency and efficiency, which we didn't able to fully optimize it.

To enhance the performance and scalability of our modified kernel:

1. Improved **getpinfo Concurrency:**
   The current **sys_getpinfo** implementation requires locking the global process table to iterate through all processes and gather statics. This is a potential performance bottleneck. We could improve this by implementing a form of **read-copy-update (RCU)** or a more granular, read only locking mechanism to minimize the time the critical process table lock is held, thereby improving overall system responsiveness**.**
2. **Dynamic Log Allocation for hapinfo:**
   The current **hapinfo** implementation use a simple fixed size array within the **struct proc.** This introduces log overflow issues for long-running processes and memory waste for process that rarely use the heap. We recommend to replace this with a dynamically allocated circular buffer or a linked list of pages t eliminate log size limitations and conserve kernel memory.
3. **Kernel API for Long Control**:
   Providing additional system calls to allow user program to **clear** and **reset** their own heap activity log would give users better control over data collection.

As, it was group project we have worked in a group consisting of total three member, and I am really proud and grateful to all my group member as they have worked really hard to complete this project and make this project successful. All the group members have contributed equally in the sections of System Call Integration and Testing, Kernel structure Modification and Init and also in the section of Core Kernel Implementation.
We are also grateful to our honorable course instructor Md. Salman Shamil, for providing us with the opportunity to work on this project. His guidance and support have been invaluable throughout development, enabling us to deepen our understanding of Operating System Design.