Computer Science 330
Operating Systems
Siena University
Fall 2025

Lab 8: Working Set Simulation

Due: 4:00 PM, Friday, December 5, 2025

In this lab, we will use a simulator to gain a better understanding of the working set frame allocation policy. There is also an opportunity to implement your own simulator for bonus programming project points.

You may work individually or in a groups of two or three.

Learning goals:

1. To gain a better understanding of how the working set model for memory frame allocation works
2. To see how a series of machine-code-like instructions generates a page reference string
3. To study the behavior of a particular reference string with the working set model, and relate that behavior back to the program that generated that string of references

AI assistance in code development is permitted but must be documented thoroughly. This must be more than just a "Copilot helped" comment; describe clearly which AI tool(s) you used, how you interacted with them, and what portions of your submitted work includes code that was all or in part generated with AI assistance.

Getting Set Up

In Canvas, you will find a link to follow to set up your GitHub repository, which will be named `ws-lab-yourgitname`, for this lab. Only one member of the group should follow the link to set up the repository on GitHub, then others should request a link to be granted write access.

Answers to written questions may be given in a PDF document committed and pushed to your repository (give the name in the `README.md` file), by writing them in a readable (reasonably nicely formatted, not all one big line of text) GitHub Markdown form in your repository's `README.md` file, or by linking to a shared document containing your answers from your `README.md` file.

Page Reference String

In order to investigate the working set allocation scheme through simulation, we use a *page reference string* that represents a sequence of memory accesses, and simulate what would happen in a working set allocation given that access string.

We will consider a "pseudo-realistic" page reference string that corresponds to the following program segment, written in a C-like language:

```
const int n=10;
int i, j, A[n], B[n], C[n], temp;

for (i=1; i<=n; i++) {
  A[i]=i;
  B[i]=n-i+1;
}
for (i=1; i<=n; i++) {
  temp=0;
  for (j=i; j<=n; j++) {
    temp=temp+A[n+i-j]*B[j];
  }
  C[i]=temp;
}
```

Using a hypothetical machine with registers denoted by Ri and a fixed instruction size of 1 word per instruction, the machine language version of this program is loaded in virtual address space (with page size 4K, *i.e.*, 1024 words) as follows:

```
0x2FBC  (R1) <- ONE            Index i
0x2FC0  (R2) <- n              Loop bound
0x2FC4  compare R1,R2          Test i>n
0x2FC8  branch_greater + 0x20
0x2FCC  A(R1) <- (R1)          Compute A[i]
0x2FD0  (R0) <- n              Compute B[i]
0x2FD4  (R0) <- (R0) - (R1)
0x2FD8  (R0) <- (R0) + ONE
0x2FDC  B(R1) <- (R0)
0x2FE0  (R1) <- (R1) + ONE     Increment i
0x2FE4  branch * - 0x20
0x2FE8  (R1) <- ONE            Index i
0x2FEC  (R2) <- n              Loop bound
0x2FF0  compare R1,R2          Test i>n
0x2FF4  branch_greater + 0x50
0x2FF8  (R0) <- ZERO           temp <- 0
0x2FFC  temp <- (R0)
0x3000  (R3) <- (R1)           Index j
0x3004  (R4) <- n              Loop bound
0x3008  compare R3,R4          Test j>n
0x300C  branch_greater + 0x20
0x3010  (R0) <- n              Compute A[n+i-j]
0x3014  (R0) <- (R0) + (R1)
0x3018  (R0) <- (R0) - (R3)
```

```
0x301C  (R5) <- A(R0)
0x3020  (R6) <- B(R3)          Compute B[j]
0x3024  (R5) <- (R5) * (R6)
0x3028  (R5) <- (R5) + temp
0x302C  temp <- (R5)
0x3030  (R3) <- (R3) + ONE     Increment j
0x3034  branch - 0x20
0x3038  C(R1) <- (R5)          Compute C[i]
0x303C  (R1) <- (R1) + ONE     Increment i
0x3040  branch - 0x50
  ...
0x6000  Storage for C
0x7000  Storage for ONE
0x7004  Storage for n
0x7008  Storage for temp
0x700C  Storage for ZERO
0x8000  Storage for A
0x9000  Storage for B
```

Upon execution of this program segment, the following reference string is generated:

ω= 272722(28272272927222)^{n}272722(272733733(373338393373737333)$^{n-i+1}$3637322)^n

Question 1: State the instruction or variable reference that causes each of the first 20 page references (up to the end of the first parenthesized portion of the reference string). (5 points)

Question 2: How long should this string be for $n = 10$? (2 points)

Question 3: For values of Δ of 2, 5, and 10, show the working set after each of the first 20 page references. Assuming only pages in the working set are kept in physical memory, indicate which page references would result in a page fault and how many page faults are generated for each Δ value. (10 points)

Working Set Simulator

In order to explore the run-time behavior of the working set memory management policy for such a reference string for different values of n and Δ, we can write a simulator.

You may choose to use my simulator (which is available as an executable on noreaster in `/home/cs330/ws/sim`) or you may choose to implement your own simulator for 20 bonus programming project points. Even if you are not going to implement your own simulator, read over this section carefully. You need to understand this description to be able to do the simulation study and analysis in the next section.

If you wish to receive the bonus points, write your simulator in the repository for this lab, and commit and push it by close of business on the last day of the semester.

The simulator will print values:

$$\begin{aligned} \Delta &= \text{window size} \\ P(\Delta) &= \text{total number of page faults} \\ W(\Delta) &= \text{average working set size} \\ F(\Delta) = \frac{P(\Delta)}{|\omega|} &= \text{average page fault rate} \end{aligned}$$

The main program takes the value of Δ as a command-line parameter. This will allow you to write a script (in your favorite scripting language) that runs the program repeatedly for the values of Δ required. The value of Δ is specified with the `-d` flag. A debugging mode can be turned on by `-D`, which shows how the working set changes as each page is referenced. The program also takes a flag `-n` to specify n in the reference string used. The default is 10, and you may use that to generate your plots. You are encouraged to try other values of n, but you need only plot for $n = 10$.

Note that as each entry in the reference string (page) is processed, one of four things will happen to the working set. (i) the page is added to the set, and none is removed, (ii) the page is added to the set and one old page is removed, (iii) the page is already in the set and another page is removed, or (iv) the page is already in the set and no other page is removed.

Our program will hard-code the reference string (actually, generate it on the fly) but the design should be modular enough that you could replace one function to generate a different reference string. The code used by the provided simulator to generate the reference string is included in the files `rs_gen.[ch]` in your repository for this lab.

A Brief Simulation Study

Use the simulator to generate the data for, and then plot the following curves:

1. Δ vs. $P(\Delta)$,
2. Δ vs. $W(\Delta)$, and
3. Δ vs. $1/F(\Delta)$,

for $n = 10$ and all values of Δ ranging from 1 to 200.

Question 4: Include these plots in your repository. (15 points)

Question 5: From the plot of Δ vs. $1/F(\Delta)$, explain the cause of all knees in the graph in terms of program (or reference string) structure. (10 points)

Question 6: Is the strategy used by this program one that could be used by a real system to keep track of a process' working set? Why or why not? (3 points)

Submission

Commit and push!

Those wishing to submit a simulation program for bonus programming project credit should tag the instructor in a GitHub Issue in the repository to request grading.

Grading

This assignment will be graded out of 45 points.

Feature	Value	Score
Question 1	5	
Question 2	2	
Question 3	10	
Question 4: plots	15	
Question 5	10	
Question 6	3	
Total	45	

Bonus simulator submissions will be graded on correctness, style, and documentation, earning up to 20 programming project points. Important note: group submissions must clearly indicate the contributions of all group members.