Lab 7 – Memory Management

Due: 9:55 AM, Thursday, April 21, 2005

This week's lab consists of practice questions to look at on your own (not to be turned in) and questions to be submitted. Your answers should be submitted in a PDF file `lab7.pdf`.

Practice Questions

- A computer system has enough room to hold four programs in its main memory. These programs are idle waiting for I/O half the time. What fraction of the CPU time is wasted?
- SG&G 8.3, 8.7, 8.9, 8.10, 8.11, 9.4, 9.12

Lab Questions

Prepare written answers to the following questions and include them in your submitted PDF file.

1. SG&G 9.10 (2 points)
2. SG&G 9.14 (2 points)
3. SG&G 9.15 (2 points)
4. Read Chapter 3 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. This chapter describes hardware support for memory management in the 32-bit Intel processor family. It is heavy reading, but don't get bogged down in too much of the detail, just look for applications of the concepts we talked about in class. The document may be found in `/home/faculty/terescoj/-shared/cs432/labs/lab7/intel-manual.pdf` or at `http://developer.intel.com/design/pentium4/manuals/index_new.htm`. **Important: please don't print this whole document. It's over 800 pages!** You need only look at Chapter 3, and maybe you can look at it electronically.

   Again, don't get bogged down in too much detail, but describe in a few paragraphs how IA-32 processors support the memory management concepts we have been discussing. (6 points)

Working Set Simulator Consider 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 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:

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

In `/home/faculty/terescoj/shared/cs432/labs/lab7` you will find a C++ program that simulates the run-time behavior of this program segment when a working set memory management policy is used. The program prints 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 given main program takes the value of Δ as a command-line parameter. This allows 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 is turned on by `-D`. 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.

1. Use this program to plot the following curves: Δ vs. $P(\Delta)$, Δ vs. $W(\Delta)$, Δ vs. $1/F(\Delta)$, for Δ ranging from 1 to 200. (6 points)

2. 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. (5 points)

3. 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? (2 points)