

Problem Set #5.**Due: March 12 before class.**

1. Certain algorithms that work very well when all of the data fit in memory do not work so well when the virtual memory exceeds the available physical memory, due to the amount of time that may be spent on page replacement. The following program multiplies two matrices, where each matrix is stored as a two dimensional array. The array is stored in row-order, meaning that consecutive elements in the same row of the array will be stored in consecutive locations in memory.

```
multiply(A,B,C)
{
    int A[SIZE][SIZE], B[SIZE][SIZE], C[SIZE][SIZE];
    int i,j,k,temp;
    for all i /* this done in parallel */
    for (j=0; j<SIZE; j++) {
        temp = 0;
        for (k=0; k<SIZE; k++)
            temp += a[i][k] * b[k][j];
        c[i][j] = temp;
    }
}
```

Suppose the above program was written for a shared-memory multiprocessor. Each processor has a number assigned to it to use for the variable *i*. Therefore, each processor is responsible for computing a row of *C*. Furthermore, assume that they all execute in parallel at the same rate. We will be interested in the performance of this program in light of very large arrays, which will not all fit in memory at the same time. For this problem, assume an optimal page replacement algorithm, where 1000 pages are available for the arrays (you can ignore *i,j,k* and the source program). Each page may hold up to 1000 elements of any array.

Assume there are 100 processors, assigned numbers from 0 to 99. Each will be responsible for doing a set of rows in the multiplication. For example, when multiplying 1000x1000 matrices, Processor 0 will find the first 10 rows, Processor 1 the next 10, and so on. The code each processor runs will be as follows:

```
int A[SIZE][SIZE], B[SIZE][SIZE], C[SIZE][SIZE];
multiply(A,B,C){
    int i,j,k,temp;
    int mult = SIZE / 100;
    for (i=mult*ProcN ; i<mult * (ProcN + 1); i++)
    for (j=0; j<SIZE; j++){
        temp = 0;
        for (k=0; k<SIZE; k++)
            temp += a[i][k] * b[k][j];
    }
```

```

    c[i][j] = temp;
  }
}

```

It is assumed that these are running roughly in parallel, though their memory accesses will be serialized by the memory unit. Therefore, accesses to the arrays will look something like this:

```

{ a[0][0], a[10][0], a[20][0], .... a[990][0] }
{ b[0][0], b[ 0][0], b[ 0][0], .... b[ 0][0] }
{ a[0][1], a[10][1], a[20][1], .... a[990][1] }
{ b[1][0], b[ 1][0], b[ 1][0], .... b[ 1][0] }

```

etc.

An Optimal Page Replacement Policy is one that accomplishes all this in the fewest page faults – based on clairvoyantly predicting what will be needed soon, and should not be replaced. Here, for example, several pages of A are used repeatedly, so should hang about in memory for a long time.

The circumstances to consider are the following:

SIZE = 1,000 (total memory required is 3 times that available)

SIZE = 10,000 (total memory required 300 times that available)

Estimate how many page faults occur in each case, (one significant digit is enough) and identify where most of them come from.

2. Assuming the page size of 4 KB and that the page table entry takes 4 bytes, how many levels of page tables would be required to map a 64-bit address space if each page table fits into a single page?
3.
 - a. What is "swap space"?
 - b. Swap space can be allocated as a special region (partition) on the disk or it could be a special file. Describe the advantages and disadvantages of these two approaches. Be sure to discuss issues related to performance, functionality, and cleanliness of design.
4. Achieving fast file reads:
 - a. Give at least three strategies that a file system can employ to reduce the time a program spends waiting for data reads to complete.
 - b. For each strategy you listed, describe a read pattern for which the strategy would do well, and one for which the strategy would do poorly.
5. Memory:
 - a. At one time, virtual memory system designers were advised to bias page replacement algorithms against modified pages in favor of those that have not been modified. The result of this suggestion was the unfortunate behavior of program code pages (which tend to be some of the only pages that are not modified) being kicked out of memory before other pages. Describe the rationale for the original suggestion of paging unmodified pages first.

- b. Explain how adding segmentation to a pure paging system can reduce the size of the page tables.
 - c. Would it ever make sense for two processes to share the same page table?
 - d. Can a first fit memory allocator ever have less fragmentation than a best fit one?
6. Consider the organization of a UNIX file as represented by the inode. Assume there are 16 direct block pointers and a singly, doubly and triply indirect pointer in each inode. Further, assume that the system block size and the disk sector size are both 4K. If the disk block pointer is 32 bits, with 2 bits to identify the physical disk and 30 bits to identify the physical block, then:
 - a. What is the maximum file size supported by this system?
 - b. What is the maximum file system partition supported by this system?
 - c. Assuming no information other than the file inode is already in main memory, how many disk accesses are required to access the byte in position Y^3+100 , where $Y=1024$?
7. Problem 11.14 Silberschatz.
8. Problem 11.12 Silberschatz.