COSC 3360 - Operating Systems Spring 2024

Programming Assignment 3 - Virtual Memory Manager

Due Date: Friday, April 26, 2024, 11:59pm CDT

In this assignment, you will design and implement a virtual memory manager with paging, and then simulate its operation for each of the following page replacement algorithms: LIFO (last-in-first-out or stack), MRU (most recently used), LRU-X (replace the page whose K-th most recent access is furthest in the past; for example, LRU-1 is simply LRU while LRU-2 replaces pages according to the time of their penultimate access; LRU-K improves significantly on LRU with regard to locality in time), LFU (least frequently used), OPT-lookhead-X (optimum with lookahead of X future addresses), and WS (Working Set). There will be 6 sets of output from these 6 runs.

The information provided here is intended to guide you in your design; it is not a complete description of all issues that must be considered. The assignment requires a combination of Unix/Linux processes (such as the page fault handler, disk driver, and page replacement process) synchronized by Unix/Linux semaphores, and simulators (such as the paging disk).

Data Structures:

You may add additional fields to these data structures if needed in your design. There is one page table per address space (process), and one table entry per page. The address can thus be divided into 2 parts: page number and displacement (offset).

A single frames table contains the address space and page number of the page currently occupying each page frame. The format of the entry for frame f is:

```
FRAMES[f]:   a p  forward_link  backward_link
```

Entries in the frames table are linked together in allocated and free lists. The disk page table entry, DPT[a,p], contains the address on the paging disk of page p of address space a.

Key Elements of the Assignment:

1. Write a page fault handler process that can be invoked by the interrupt dispatcher when a page fault occurs. The address space and page number of the missing page are made available to the fault handler by the addressing hardware. The fault handler requests a page transfer by placing an entry in the disk driver work queue and signaling the associated semaphore.

2. Design a disk driver process which schedules all I/O to the paging disk. disk command

   ```
   STARTDISK(read/write, memory_addr, disk_addr)
   ```

   initiates an I/O operation. The paging disk is wired to the semaphore of the disk driver process, which it signals when a command is complete. The disk has a work queue containing entries of the form

   ```
   (process_id, read/write, frame_index, disk_addr).
   ```

3. Assume that the page replacement algorithm runs as a separate process which is signaled each time a page frame is removed from the pool. the replacement process attempts to maintain a free pool size between min and max frames. To accomplish this, one or more processes may need to be "deactivated" and "activated" later.

   Note: In this assignment, you need not consider the actual creation and deletion of address spaces. Be sure to initialize the semaphores you use with correct initial counts.

Input and Output:

The input to your simulator is as follows:

```
tp    /* total_number_of_page_frames (in main memory) */
ps    /* page size (in number of bytes) */
r     /* number_of_page_frames_per_process for LIFO, MRU, LRU-K,
         LFU and OPT,
         or delta (window size) for the Working Set algorithm */
X     /* lookahead window size for OPT, X for LRU-X, 0 for others (which do
         not use this value) */
min   /* min free pool size */
max   /* max free pool size */
k     /* total number of processes */
pid1  size1  /* process_id followed by total number of page frames on disk */
pid2  size2
 :      :
 :      :
pidk  sizek
```

These parameters are followed by a list of process id and address pairs: pid addr. You need to extract the page number from the address addr. The last address accessed by process i is followed by: i -1.

the output from your simulator includes the following data. You may also show other useful statistics.

```
number_of_page_faults for each process;
for Working Set, show the min and max size of the set and
total_number_of_page_faults
```

Bonus:

In the bonus part (30% extra) of this assignment, you will incorporate the disk scheduler in the disk driver process of the above virtual memory manager. You will implement the following disk-scheduling algorithms on separate simulations: FIFO (first-in-first-out), SSTF, and SCAN. You can model/emulate the disk as an array (or linked list) and disk tracks as elements in the array. Assume each track on the disk stores one page frame. Also, to ensure a realistic simulation (the disk access is much slower than main memory), the disk queue contains y requests (from page faults requiring the reading of a page from the disk).

Input and Output:

The expanded input to your simulator is as follows:

```
tp    /* total_number_of_page_frames (in main memory) */
ps    /* page size (in number of bytes) */
r     /* number_of_page_frames_per_process for LIFO, MRU, LRU-K,
         LFU and OPT,
         or delta (window size) for the Working Set algorithm */
X     /* lookahead window size for OPT, X for LRU-X, 0 for others (which do
         not use this value) */
min   /* min free pool size */
max   /* max free pool size */
k             /* total number of processes */
maxtrack      /* (NEW) largest-numbered track on the disk */
y             /* (NEW) number of I/O requests in the disk queue */
pid1  size1  /* process_id followed by total number of page frames on disk */
0           track for page "0"       /* (NEW) track where page 0 is stored */
:             :
size1-1     track for page "size1-1" /* (NEW) track where page size1-1 is stored */
pid2  size2
0           track for page "0"       /* (NEW) track where page 0 is stored */
:             :
size2-1     track for page "size2-1" /* (NEW) track where page size2-1 is stored */
 :     :
 :     :
pidk  sizek
0           track for page "0"       /* (NEW) track where page 0 is stored */
:             :
sizek-1     track for page "sizek-1" /* (NEW) track where page sizek-1 is stored */
```

These parameters are followed by a list of process id and address pairs: pid addr. You need to extract the page number from the address addr. The last address accessed by process i is followed by: i -1.

The output from your simulator includes the following data. You may also show other useful statistics.

```
number_of_page_faults for each process;

for Working Set, show the min and max size of the set and
total_number_of_page_faults;

total and average seek times for each combination of page replacement
and disk scheduling algorithms.
```

There will be 18 sets of output from these 18 runs.