Report for CSC3150 A3

Name：桂驰
ID：120090194

Environment

OS

NAME="CentOS Linux"

VERSION="7 (Core)"

ID="centos"

ID_LIKE="rhel fedora"

VERSION_ID="7"

PRETTY_NAME="CentOS Linux 7 (Core)"

ANSI_COLOR="0;31"

CPE_NAME="cpe:/o:centos:centos:7"

HOME_URL="https://www.centos.org/"

BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"

CENTOS_MANTISBT_PROJECT_VERSION="7"

REDHAT_SUPPORT_PRODUCT="centos"

REDHAT_SUPPORT_PRODUCT_VERSION="7"

VS

版本: 1.73.0 (user setup)

提交: 8fa188b2b301d36553cbc9ce1b0a146ccb93351f

日期: 2022-11-01T15:34:06.111Z

Electron: 19.0.17

Chromium: 102.0.5005.167

Node.js: 16.14.2

V8: 10.2.154.15-electron.0

OS: Windows_NT x64 10.0.22000

沙盒化: No

CUDA

nvcc: NVIDIA (R) Cuda compiler driver

Copyright (c) 2005-2022 NVIDIA Corporation

Built on Wed_Jun__8_16:49:14_PDT_2022

Cuda compilation tools, release 11.7, V11.7.99

Build cuda_11.7.r11.7/compiler.31442593_0

GPU

03:00.0 VGA compatible controller: ASPEED Technology, Inc. ASPEED Graphics Family (rev 41) (prog-if 00 [VGA controller])

Subsystem: ASPEED Technology, Inc. ASPEED Graphics Family

Flags: bus master, medium devsel, latency 0, IRQ 17, NUMA node 0

Memory at 98000000 (32-bit, non-prefetchable) [size=64M]

Memory at 9c000000 (32-bit, non-prefetchable) [size=128K]

I/O ports at 2000 [size=128]

Expansion ROM at <unassigned> [disabled]

Capabilities: <access denied>

Kernel driver in use: ast

Kernel modules: ast

af:00.0 VGA compatible controller: NVIDIA Corporation Device 1eb1 (rev a1) (prog-if 00 [VGA controller])

Subsystem: NVIDIA Corporation Device 12a0

Flags: bus master, fast devsel, latency 0, IRQ 86, NUMA node 1

Memory at ed000000 (32-bit, non-prefetchable) [size=16M]

Memory at 3effe0000000 (64-bit, prefetchable) [size=256M]

Memory at 3efff0000000 (64-bit, prefetchable) [size=32M]

I/O ports at e000 [size=128]

[virtual] Expansion ROM at ee000000 [disabled] [size=512K]

Capabilities: <access denied>

Kernel driver in use: nvidia

Kernel modules: nouveau, nvidia_drm, nvidia

Execution Steps

Main Task

- cd Assignment_3_120090194/source
- sbatch slurm.sh

Bonus (Version 1)

- cd Assignment_3_120090194/bonus
- sbatch slurm.sh

Design Thought

Main Task

The whole structure is showed in Figure 1. (VM means virtual memory, PM means physical memory, SM means secondary memory, PN means page number for virtual memory, P_PN means page number for physical memory, CUT means last used time, v means valid value)

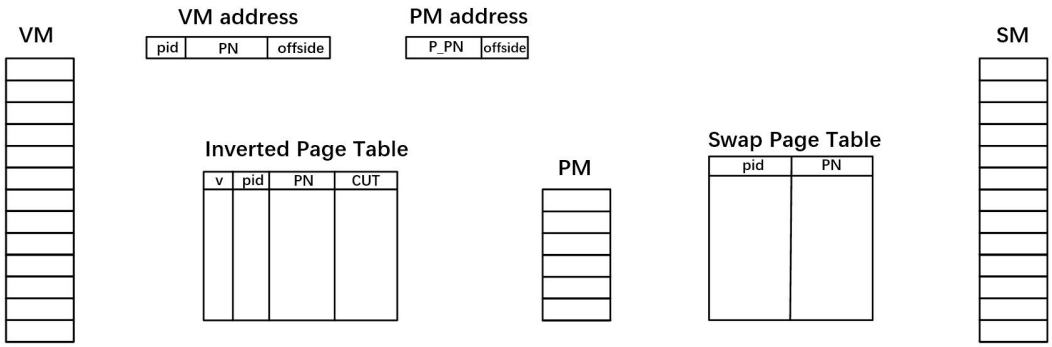


Figure 1: Whole Structure

The write or read flow char is showed in Figure 2. (Action means write or read; IPT means Inverted page table, PAD means Physical address, IE means invalid page entry, SPT means swap page table, PPN means physical page entry, PE means page entry.)

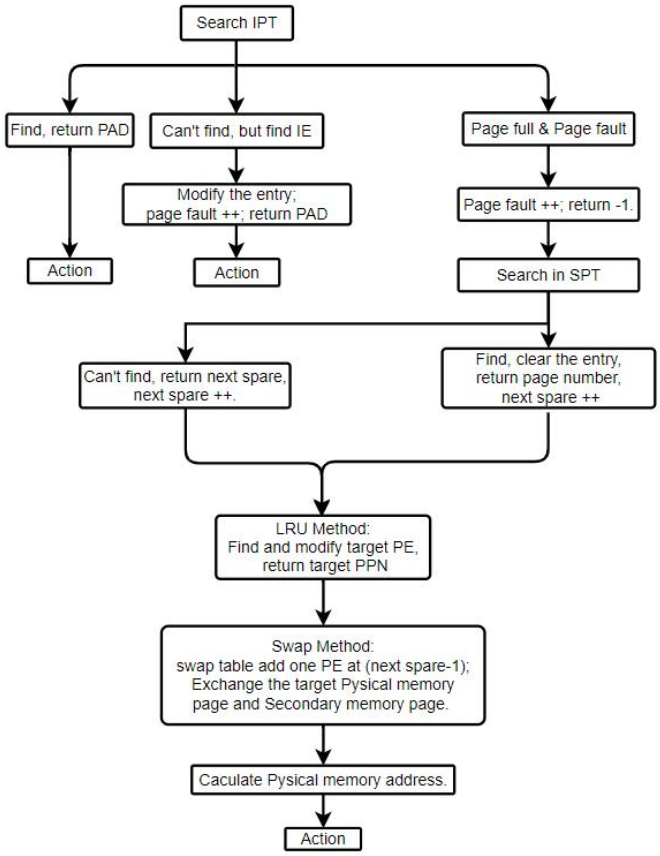


Figure 2: write or read flow chart

Inverted page table entry: a structure contains valid value (v), pid, page number, recent used time (CUT).

Swap page table entry: a structure contains pid and page number.

Search in Inverted page table or Swap page table: go through the table and compare the valid value, pid and page number.

LRU Method: go through the inverted page table, and find the smallest recent used time entry.

Swap Method: exchange two page.

Bonus (Version 1)

The bonus part has same structure and write or read flow chart. There are also some changes:
<<<1,1>>> => <<<1,4>>>
Pid: pid equal to the thread id.
Virtual memory address: every virtual memory address should divide by 4. (eg. 5/4 => 1)
LRU Method: every thread only can change the entry whose pid equals to thread id.

Page Fault and Explain

Test Case 1: 8193
Explain: user program first writes the whole data.bin file to the Physical Memory. Every address is new in. Thus, page fault = Input size / Page size = (1<<17) / (1<<5) = 4096. Second read 32kB + 1B size data from bottom to up. Current physical memory contains 96KB to 128KB data. Thus, there only one page fault. Last step, user program read all the data from top to bottom. The inverted page table updates 4 times. Thus, page fault equals to 4096.
In the end, page fault = 4096 + 1 + 4096 = 8193.
Test Case 2: 9125
Explain: user program first writes the whole data.bin file to the Physical Memory start at address 32*1024. Thus, page fault is same as test case 1, which is 4096. Second, write some data (32KB-32B) to VM starting from 0. The VM did not store the address and need to add new page. Thus, page fault = 1023. Last step, user program read all the data from 32*1024 to bottom. The inverted page table updates 4 times. Thus, page fault equals to 4096.
In the end, page fault = 4096 + 1023 + 4096 = 9125.
Bonus (Version 1): 8193
Explain: The task is divided into 4 threads to complete. Page fault for every thread:
Thread 1: 1024 + 1 + 1024
Thread 2: 1024 + 0 + 1024
Thread 3: 1024 + 0 + 1024
Thread 4: 1024 + 0 + 1024
For thread 1, it store the data and read data whose addr%4==0. The address will divide by 4. That means, address 0, 4, 8, …, 31*4 will be stored in one page. Thus, the first write part's page fault = 4096/4 = 1024. Similarly, the last part is the same (1024). For the second step, read 32KB + 1B. There exits one page fault. The other threads are similar in first step and last step. For the second step, their page fault equals to 0.

Problem and Solution

Problem: How to design inverted page table
Solution: the inverted page table is used to translate virtual memory address to physical memory address. It contains 4 elements, valid value (v), pid, page number, recent used time (CUT). And the entry index corresponds to the page number of physical memory.

Problem: How to implement LRU Method
Solution: I define a variable called time. When search in the inverted page table, time++. And the table entry contains an element called CUT which is copy by time. When we need LRU method, the program will go through whole table and find the smallest CUT, and return that entry.

Problem: How to swap
Solution: swap is to exchange the content between physical memory and secondary memory. It is worth nothing that the swap unit is page (32 Byte) instead of 1 Byte. When swap over, the program should update the swap table.

Problem: How to implement multiple threads program
Solution: I define a variable called pid, which is equals to the thread id. Every thread can only manipulate the page entry whose pid is equals to current thread id. That is, the LRU method will only return the correct page entry. Because thread can not access the physical memory of other thread.

Learning outcomes

1. **A better understanding of memory management strategies, especially for the page part.**
2. **Learn how to design inverted page table, swap page table, LRU method, swap method.**
3. **Learn how to use ssh in the vscode.**
4. **Learn how to use cuda and write simple cuda c program.**
5. **Learn how to invoke multiple threads using cuda.**
6. **Self-learning ability improvement.**
7. **C program ability improved.**

Screenshot of program output

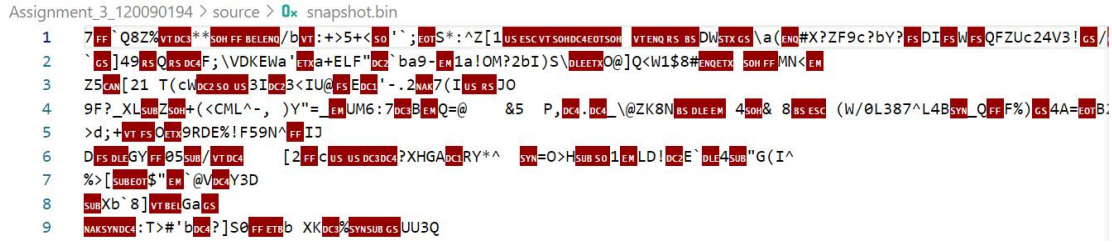
Main Task

Test Case 1

```
[120090194@node21 csc3150]$ cd /nfsmnt/120090194/csc3150/Assignment_3_120090194/source
[120090194@node21 source]$ sbatch slurm.sh
Submitted batch job 26139
```

```
 9   input size: 131072
10   pagefault number is 8193
```

Assignment_3_120090194 > source > 0x snapshot.bin

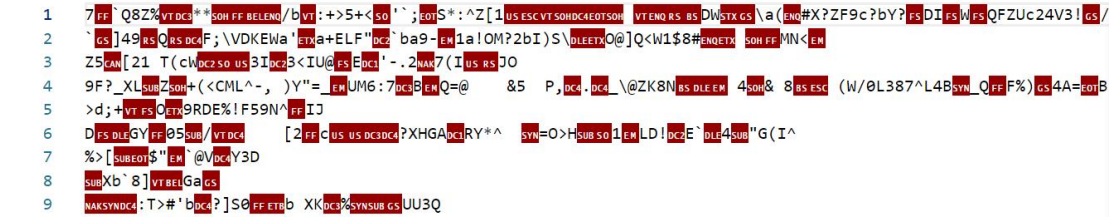


Test Case 2

```
[120090194@node21 source]$ sbatch slurm.sh
Submitted batch job 26156
```

```
 9   input size: 131072
10   pagefault number is 9215
11
```

Assignment_3_120090194 > source > 0x snapshot.bin



Bonus (Version 1)

```
[120090194@node21 csc3150]$ cd Assignment_3_120090194/bonus
[120090194@node21 bonus]$ sbatch slurm.sh
Submitted batch job 26174
```

```
 9   input size: 131072
10   pagefault number is 8193
11
```

Assignment_3_120090194 > bonus > 0x snapshot.bin