Course Code:	CE2005/CZ2005 Operating Systems
Name:	Adrian Goh Jun Wei (U1721134D)
Lab Group:	SS8
Coursework:	Lab 3 Report

int VpnToPhyPage(int vpn)

```
int VpnToPhyPage(int vpn)
{
  //your code here to get a physical frame for page vpn
  //you can refer to PageOutPageIn(int vpn) to see how an entry was created in ipt

  // DEBUG message to display all the values of a physical frame
  for (int i = 0; i < NumPhysPages; i++)
     DEBUG('p', "...IPT %i, pid, vpn, last used and valid is %i %i %i %i \n", i, memoryTable[i].pid,
memoryTable[i].vPage, memoryTable[i].lastUsed, memoryTable[i].valid);

  // Loop through to find the correct physical page by comparing pid and vpn
  // If found, return the physical frame number
  // Else, return -1
  for (int i = 0; i < NumPhysPages; i++){
        if ((memoryTable[i].valid) && (memoryTable[i].pid == currentThread->pid) && (memoryTable[i].vPage == vpn)){
                DEBUG('p', ".....Found IPT %i, pid, vpn, last used and valid is %i %i %i %i \n", i, memoryTable[i].pid,
memoryTable[i].vPage, memoryTable[i].lastUsed, memoryTable[i].valid);
                return i;
        }
  }
  DEBUG('p', ".....IPT with matching pid and vpn NOT found!\n");
  return -1;
}
```

void InsertToTLB(int vpn, int phyPage)

```
void InsertToTLB(int vpn, int phyPage)
{
  int i = 0; //entry in the TLB

  //your code to find an empty in TLB or to replace the oldest entry if TLB is full
  // value of FIFOPointer is the TLB that will be replaced next
  static int FIFOPointer = 0;

  // Using a loop to find an empty entry in TLB
  // First checks whether there is an old entry that is invalid (valid bit == 0). Iffound, bre ak and exit the loop
  // Else, FIFOPointer to point to the oldest entry that will be removed next
  bool isEmpty  = false;
  for (i = 0; i < TLBSize; i++){
        if (!machine->tlb[i].valid){
                isEmpty  = true;
                break;
        }
  }
  if (!isEmpty)
        i = FIFOPointer;

  // if an entry is just inserted, then the entry next to it is the oldest entry
  FIFOPointer = (i + 1) % TLBSize;
```

int lruAlgorithm(void)

```
int lruAlgorithm(void)
{
  //your code here to find the physical frame that should be freed
  //according to the LRU algorithm.

  // Firsty, it checks if there's an invalid frame. If so, make the invalid frame the victim page.
  // However, if all pages are valid, then LRU compares the last used information in each frame to select the victim
page
  int phyPage = 0;
  int last = memoryTable[0].lastUsed;  // last used field
  for (int i = 0; i < NumPhysPages; i++){
        if (!memoryTable[i].valid){ // if free frame is found
                phyPage = i;
                break;
        }
        if (memoryTable[i].lastUsed < last){  // no free frame found
                last = memoryTable[i].lastUsed;
                phyPage = i;
        }
  }

  return phyPage;
}
```

Experimental result table

tick	vpn	pid	IPT[0]	IPT[1]	IPT[2]	IPT[3]	TLB[0]	TLB[1]	TLB[2]	Phy Page Out
10	0	0	0,0,0,0	0,0,0,0	0,0,0,0	0,0,0,0	0,0,0	0,0,0	0,0,0	
13	9	0	0,0,12,1	0,0,0,0	0,0,0,0	0,0,0,0	0,0,1	0,0,0	0,0,0	
15	26	0	0,0,12,1	0,9,15,1	0,0,0,0	0,0,0,0	0,0,1	9,1,1	0,0,0	
20	1	0	0,0,12,1	0,9,19,1	0,26,17,1	0,0,0,0	0,0,1	9,1,1	26,2,1	
26	0	0	0,0,12,1	0,9,25,1	0,26,17,1	0,1,22,1	1,3,1	9,1,1	26,2,1	
28	10	0	0,0,28,1	0,9,25,1	0,26,17,1	0,1,22,1	1,3,1	0,0,1	26,2,1	2
41	9	0	0,0,40,1	0,9,25,1	0,10,28,1	0,1,22,1	1,3,1	0,0,1	10,2,1	
42	26	0	0,0,40,1	0,9,42,1	0,10,28,1	0,1,22,1	9,1,1	0,0,1	10,2,1	
47	0	0	0,0,40,1	0,9,46,1	0,10,28,1	0,26,44,1	9,1,1	26,3,1	10,2,1	
59	0	1	0,0,49,1	0,9,46,1	0,10,28,1	0,26,44,1	9,1,0	26,3,0	0,0,0	
62	9	1	0,0,49,1	0,9,46,1	1,0,61,1	0,26,44,1	0,2,1	26,3,0	0,0,0	3
64	26	1	0,0,49,1	0,9,46,1	1,0,61,1	1,9,64,1	0,2,1	9,3,1	0,0,0	
69	1	1	0,0,49,1	1,26,66,1	1,0,61,1	1,9,68,1	0,2,1	9,3,1	26,1,1	
74	0	1	1,1,71,1	1,26,66,1	1,0,61,1	1,9,73,1	1,0,1	9,3,1	26,1,1	
117	0	0	1,1,71,1	1,26,66,1	1,0,76,0	1,9,73,0	1,0,0	0,2,0	26,1,0	
120	9	0	0,0,119,1	1,26,66,0	1,0,76,0	1,9,73,0	0,0,1	0,2,0	26,1,0	
122	10	0	0,0,119,1	0,9,121,1	1,0,76,0	1,9,73,0	0,0,1	9,1,1	26,1,0	
123	26	0	0,0,119,1	0,9,121,1	0,10,123,1	1,9,73,0	0,0,1	9,1,1	10,2,1	
125	0	0	0,0,119,1	0,9,121,1	0,10,124,1	0,26,124,1	26,3,1	9,1,1	10,2,1	

Legends:

IPT entry	:	(pid, vpn, lastUsed, valid)
TLB entry	:	(vpn, phy, valid)

Experimental details and screenshot of the summarized output

```
Ticks: total 127, idle 0, system 70, user 57
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 14, outs 2, tlb miss: 19
Network I/O: packets received 0, sent 0
```

Page size (defined in NachOS)	128 bytes
Number of physical frames (defined in NachOS)	4
TLB size (defined in NachOS)	3
Number of pages used by the test program	5
Number of TLB miss	19
Number of page faults	14
Number of page out	2

Analysis of the test program output

To aid in analysis, DEBUG commands are added

DEBUG code in the function *VpnToPhyPage()*

```
// DEBUG message to display all the values of a physical frame
for (int i = 0; i < NumPhysPages; i++)
    DEBUG('p', "...IPT %i, pid, vpn, last used and valid is %i %i %i %i \n", i, memoryTable[i].pid,
memoryTable[i].vPage, memoryTable[i].lastUsed, memoryTable[i].valid);
```

DEBUG code in the function *InsertToTLB()*

```
 // to show the TLB and their respective values
DEBUG('p', "Before updating TLB %i \n", i);
  for (int j=0; j<3; j++)
    DEBUG('p', "...TLB %i, vpn, phy and valid is %i %i %i \n", j, machine->tlb[j].virtualPage,
machine->tlb[j].physicalPage, machine->tlb[j].valid);
```

Tick	Explanation of events happening
10	The valid bits in all the IPTs and TLBs are set as 0 and thus, invalid. This causes a TLB miss and a page fault in IPT. Thus, the required frame is paged in and updated in both TLB and IPT. IPT[0] - Updated with pid = 0, vpn = 0, valid bit = 1 TLB[0] - Updated with vpn = 0 and phyPage = 0 - From [0,0,0] to [0,0,1]
13	Events happening is similar to Tick 10 IPT[1] - Updated with pid = 0 and vpn = 9 TLB[1] - Updated with vpn = 9 and phyPage = 1 - From [0,0,0] to [9,1,1]
15	Events happening is similar to Tick 10 IPT[2] - Updated with pid = 0 and vpn = 26 TLB[2] - Updated with vpn = 26 and phyPage = 2 - From [0,0,0] to [26,2,1]
20	The program checks if vpn 1 is present in the TLB, but encounter TLB miss as it is not found. Since IPT[3] is empty, the required page has to be paged in and both the TLB and IPT have to be updated. However, since TLB table is fully occupied, FIFO replacement occurs and the oldest entry TLB[0] is updated. IPT[3] - Updated with pid = 0 and vpn = 1

	TLB[0] - Updated with vpn = 1 and phyPage = 3 - From [0,0,1] to [1,3,1]
26	The program checks if vpn 0 is present in the TLB, but encounter TLB miss as it is not found. However, it can be found in IPT[0] and thus, used to update the TLB. TLB[1] is updated based on FIFO policy as it is the oldest entry TLB[1] - Updated with vpn = 0 and phyPage = 0 - From [9,1,1] to [0,0,1]
28	The program checks if vpn 10 is present in TLB and IPT, but encounter TLB miss and page fault as it is not found. The required entry has to be paged in. As all IPT are occupied, IPT[2] will be updated under LRU as it has the smallest lastUsed values amongst the other IPT. As IPT[2] is modified, it is a “dirty” page and thus, paged out and replaced. In addition, according to the FIFO policy, the oldest entry TLB[2] will be replaced. IPT[2] - Updated with pid = 0 and vpn = 10 TLB[1] - Updated with vpn = 10 and phyPage = 2 - From [26,2,1] to [10,2,1]
41	Events happening is similar to Tick 26. The program checks if vpn 9 is present in the TLB, but encounter TLB miss as it is not found. However, it can be found in IPT[1] and thus, used to update the TLB. TLB[0] is updated based on FIFO as it is the oldest entry TLB[0] - Updated with vpn = 9 and phyPage = 1 - From [1,3,1] to [9,1,1]
42	The program checks if vpn 26 is present in TLB and IPT, but encounter TLB miss and page fault as it is not found. IPT[3] will be replaced based on LRU, while TLB[1] will be replaced based on FIFO IPT[3] - Updated with pid = 0 and vpn = 26 TLB[1] - Updated with vpn = 26 and phyPage = 3 - From [0,0,1] to [26,3,1]
47	Events happening is similar to Tick 26. The program checks if vpn 0 is present in the TLB, but encounter TLB miss as it is not found. However, it can be found in IPT[0] and thus, used to update the TLB. TLB[2] is updated based on FIFO as it is the oldest entry TLB[2] - Updated with vpn = 0 and phyPage = 0 - From [10,2,1] to [0,0,0]

59	A new process (pid = 1) is running. TLB with vpn 0 is being searched and as no match is found, it leads to TLB miss and subsequently Page fault. Since all TLB entries are now invalid, the 1st entry TLB[0] will be used to store the new entry. IPT[2] will be replaced by the new entry being paged in due to LRU IPT[2] - Updated with pid = 1 and vpn = 0 TLB[0] - Updated with vpn = 0 and phyPage = 2 - From [9,1,0] to [0,2,1]
62	Pid 1 is still running. Subsequently, it is similar to Tick 28. The program checks if vpn 9 is present in TLB and IPT, but encounter TLB miss and page fault as it is not found. The required entry has to be paged in. As all IPT are occupied, IPT[3] will be updated under LRU as it has the smallest lastUsed values amongst the other IPT. As IPT[3] is modified, it is a “dirty” page and thus, paged out and replaced. In addition, according to the FIFO policy, the oldest entry TLB[1] will be replaced. IPT[3] - Updated with pid = 1 and vpn = 9 TLB[1] - Updated with vpn = 9 and phyPage = 3 - From [26,3,0] to [9,3,1]
64	Pid 1 is still running. The program checks if vpn 26 is present in TLB and IPT, but encounter TLB miss and page fault as it is not found. As TLB[2] is still invalid, it will be used to store the new entry. IPT[1] will be replaced based on LRU. . IPT[1] - Updated with pid = 1 and vpn = 26 TLB[2] - Updated with vpn = 26 and phyPage = 1 - From [0,0,0] to [26,1,1]
69	Pid 1 is still running. The program checks if vpn 1 is present in TLB and IPT, but encounter TLB miss and page fault as it is not found. TLB[0] and IPT[0] will be replaced using FIFO and LRU respectively. IPT[0] - Updated with pid = 1 and vpn = 1 TLB[0] - Updated with vpn = 1 and phyPage = 0 - From [0,2,1] to [1,0,1]
74	Pid 1 is still running. The program searches for a TLB with vpn 0. Since it cannot be found in TLB, TLB miss occurs. However, as IPT[2] has “vpn 0”, page hit occur and the required entry will be fetched from IPT[2].

	TLB[1] - Updated with vpn = 0 and phyPage = 2 - From [9,3,1] to [0,2,0]
117	Now the main thread is running and all the valid bits of the entries in both IPT and TLB are set as 0. It causes a TLB miss in TLB, followed by Page fault in IPT. Therefore, the required frame should be paged in and updated accordingly in both TLB and IPT. IPT[0] - Updated with pid = 0 and vpn = 0 TLB[0] - Updated with vpn = 0 and phyPage = 0 - From [1,0,0] to [0,0,1]
120	Events happening is similar to Tick 117, but for vpn 9. IPT[1] - Updated with pid = 0 and vpn = 9 TLB[1] - Updated with vpn = 9 and phyPage = 1 - From [0,2,0] to [9,1,1]
122	Events happening is similar to Tick 117, but for vpn 10 IPT[2] - Updated with pid = 0 and vpn = 10 TLB[2] - Updated with vpn = 10 and phyPage = 2 - From [26,1,0] to [10,2,1]
123	The program checks if vpn 26 is present in the TLB, but encounter TLB miss as it is not found. Since IPT[3] is empty, the required page has to be paged in and both the TLB and IPT have to be updated. However, since TLB table is fully occupied, FIFO replacement occurs and the oldest entry TLB[0] is updated. IPT[3] - Updated with pid = 0 and vpn = 26 TLB[0] - Updated with vpn = 26 and phyPage = 3 - From [0,0,1] to [26,3,1]
125	Events happening is similar to Tick 74, but running on pid 0 instead of pid 1. The program searches for a TLB with vpn 0. Since it cannot be found in TLB, TLB miss occurs. However, as IPT[2] has “vpn 0”, page hit occur and the required entry will be fetched from IPT[2] to update TLB[1] due to FIFO. TLB[1] - Updated with vpn = 0 and phyPage = 0