Quản lý bộ nhớ

# Bài tập 1 : Xem thông tin bộ nhớ

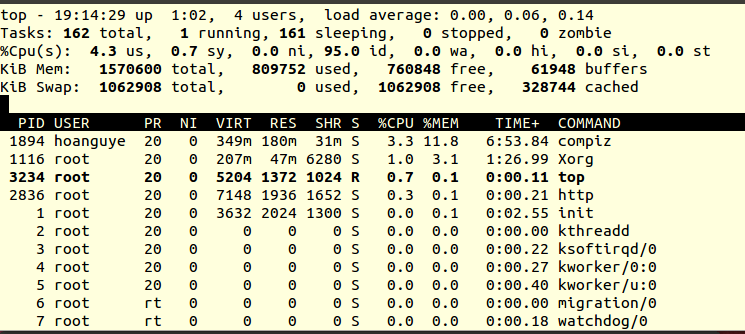
1. Sử dụng **top, ps** đọc thông tin về kích thước vùng nhớ của 1 tiến trình ?
2. Tìm hiểu các thông tin hiển thị bởi các lệnh sau ?

**free vmstat**

**cat /proc/meminfo**

1. Tìm hiểu lệnh trên Windows cung cấp thông tin về bộ nhớ của tiến trình ?

## Tham khảo lệnh top :



**Thông tin chung :**

* + the time your system is been up
  + number of users logged in
  + load average of 5, 10 and 15 minutes respectively.



## Thống kê về các task và process hiện hành:



**Thống kê về trạng thái của cpu :**

* + - us, user: CPU time in running (un-niced) user processes
    - sy, system: CPU time in running kernel processes
    - ni, niced: CPU time in running niced user processes
    - wa, IO wait: CPU time waiting for IO completion
    - hi: CPU time serving hardware interrupts
    - si: CPU time serving software interrupts
    - st: CPU time stolen for this vm by the hipervisor.



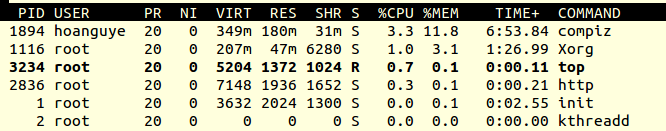
## Thống kê về trạng thái của memory :

Thông tin về physical memory : total available memory, used memory, free memory, and memory used for buffers

Thông tin về swap space : total available memory, used memory, free memory, and memory used for cached



## Danh sách các task và process : mỗi task /process được hiển thị trên một dòng , bao gồm các thông tin:



**PID**

The Process ID, to uniquely identify a processes.

## USER

The effective user name of the owner of the processes.

## PR

The scheduling priority of the process. Some values in this field are ‘rt’. It means that the process is running under real-time.

## NI

The nice value of the process. Lower values mean higher priority.

## VIRT

The amount of virtual memory used by the process.

## RES

The resident memory size. Resident memory is the amount of non-swapped physical memory a task is using.

## SHR

SHR is the shared memory used by the process.

## S

This is the process status. It can have one of the following values: D - uninterruptible sleep

R - running S - sleeping

T - traced or stopped Z - zombie

## %CPU

It is the percentage of CPU time the task has used since last update.

## %MEM

Percentage of available physical memory used by the process.

## TIME+

The total CPU time the task has used since it started, with precision upto hundredth of a second.

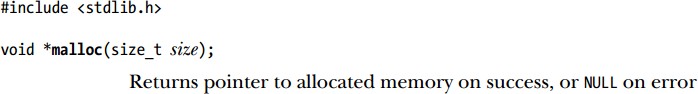
## COMMAND

The command which was used to start the process.

# Bài tập 2 : cấp phát bộ nhớ

Heap là vùng nhớ thuộc không gian địa chỉ của tiến trình, có kích thước biến động. Kích thước Heap có thể tăng hay thu nhỏ khi tiến trình cấp phát hay giải phóng vùng nhớ.

Tiến trình có thể dùng hàm **malloc()** để cấp một số bytes từ Heap. Hàm malloc() trả về một con trỏ tới điểm bắt đầu của vùng nhớ cấp phát.



Tiến trình cũng sử dụng hàm **free()** để giải phóng vùng nhớ đã cấp trước đó bằng **malloc(),** khi không cần dùng.



**Đọc, hiểu và thực thi chương trình sau ?**

#include <stdio.h> #include <stdlib.h> int main(){

int n,i,\*ptr,sum=0;

printf("Enter number of elements: "); scanf("%d",&n);

ptr=(int\*)malloc(n\*sizeof(int)); //memory allocated using malloc if(ptr==NULL)

{

printf("Error! memory not allocated."); exit(0);

}

printf("Enter elements of array: "); for(i=0;i<n;++i)

{

scanf("%d",ptr+i); sum+=\*(ptr+i);

}

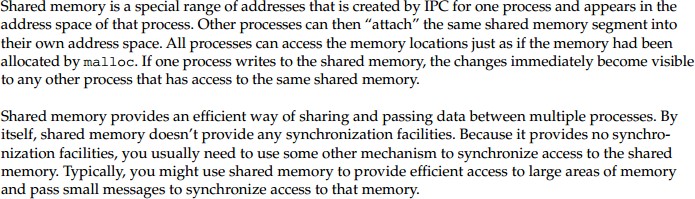
printf("Sum=%d",sum); free(ptr);

return 0;

}

# Bài tập 3 : Chia sẻ vùng nhớ chung

Hai tiến trình độc lập với nhau có thể chia sẽ dữ liệu qua một vùng nhớ chung (shared memory). Đọc một mô tả về shared memory :



Một tiến trình tạo một shared memory segment sử dụng **shmget()**. Tiến trình tạo segment này có thể gán quyền (hoặc hủy bỏ quyền đã gán) cho một tiến trình khác (user khác) bằng sử dụng **shmctl()**. Một tiến trình truy suất segment này bằng cách attached vùng nhớ vào không gian địa chỉ của nó , sử dụng hàm **shmat()** . Tiến trình cũng có thể detached segment này khỏi không gian địa chỉ của nó , sử dụng **shmdt().** Khi đã attached, tiến trình có thể đọc/ghi vào segment tùy thuộc vào quyền hạn.

#include <sys/ipc.h> #included <sys/shm.h>

**int shmget(key\_t key, size\_t size, int shmflg);**

Nếu thành công, hàm trả về shared memory segment ID , thường gọi là shmid ; trả về -1 nếu lỗi Các tham số :

* key : là một giá trị (tên) gán cho shared memory segment
* Size : là kích thước bằng byte của shared memory segment
* shmflg : chỉ định quyền truy suất, có thể là “SHM\_R”, “SHM\_W”, “SHM\_R| SHM\_W”

**void \*shmat(int shmid, const void \*shmaddr, int shmflg);**

// shmat() trả về một con trỏ , shmaddr , trỏ tới vị trí đầu của shared segment.

**int shmdt(const void \*shmaddr);**

Hai chương trình trong minh họa sau chia sẽ một chuỗi qua vùng nhớ chung :

**shm-server.c** : tạo một vùng nhớ chung và ghi vào đó một chuỗi

**shm\_client.c** : attaches với vùng nhớ chung được tạo bởi chương trình shm\_server và in chuỗi đọc được từ vùng nhớ

## Đọc, hiểu và thực thi 2 chương trình để thấy kết quả ?

**shm\_server.c**

// [www.cs.cf.ac.uk/Dave](http://www.cs.cf.ac.uk/Dave) #include <sys/types.h> #include <sys/ipc.h> #include <sys/shm.h> #include <stdio.h>

#define SHMSZ 27

main()

{

char c; int shmid; key\_t key;

char \*shm, \*s;

**/\* We'll name our shared memory segment "5678". \*/**

key = 5678;

**/\* Create the segment. \*/**

if ((shmid = shmget(key, SHMSZ, IPC\_CREAT | 0666)) < 0) { perror("shmget");

exit(1);

}

**/\* Now we attach the segment to our data space. \*/**

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) { perror("shmat");

exit(1);

}

**/\*Now put some things into the memory for the other process to read. \*/**

s = shm;

for (c = 'a'; c <= 'z'; c++)

\*s++ = c;

\*s = NULL;

**/\* Finally, we wait until the other process changes the first character of our memory to '\*', indicating that it has read what we put there. \*/**

while (\*shm != '\*') sleep(1);

exit(0);

}

**shm\_client.c**

/\* shm-client - client program to demonstrate shared memory \*/ #include <sys/types.h>

#include <sys/ipc.h> #include <sys/shm.h> #include <stdio.h>

#define SHMSZ

main()

{

int shmid; key\_t key; char \*shm, \*s;

**/\* We need to get the segment named "5678", created by the server. \*/**

key = 5678;

**/\* Locate the segment \*/**

if ((shmid = shmget(key, SHMSZ, 0666)) < 0) { perror("shmget");

exit(1);

}

**/\* Now we attach the segment to our data space. \*/**

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) { perror("shmat");

exit(1);

}

**/\* Now read what the server put in the memory. \*/**

for (s = shm; \*s != NULL; s++) putchar(\*s);

putchar('\n');

**/\* Finally, change the first character of the segment to '\*', indicating we have read the segment. \*/**

\*shm = '\*';

exit(0);

}