แก้ไขครั้งล่าสุดเมื่อวันที่ 4 กันยายน 2567

Process outline

What is process?

Process in memory, process state, process control block, threads.

Process scheduling: scheduling queue, schedulers.

Operations on processes: process creation, process termination.

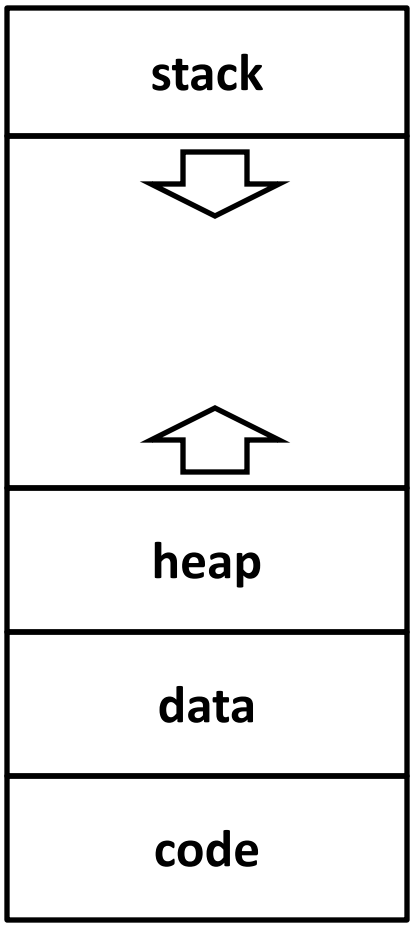
Interprocess communication: shared memory, ~~message passing~~.

Client-server systems: sockets, ~~remote procedure calls~~, pipes.

Homework: Server/Client programming

Process

A process (job) is an executable program which is submitted to OS.
OS manages the process from its birth to death (error handling).



Process in memory

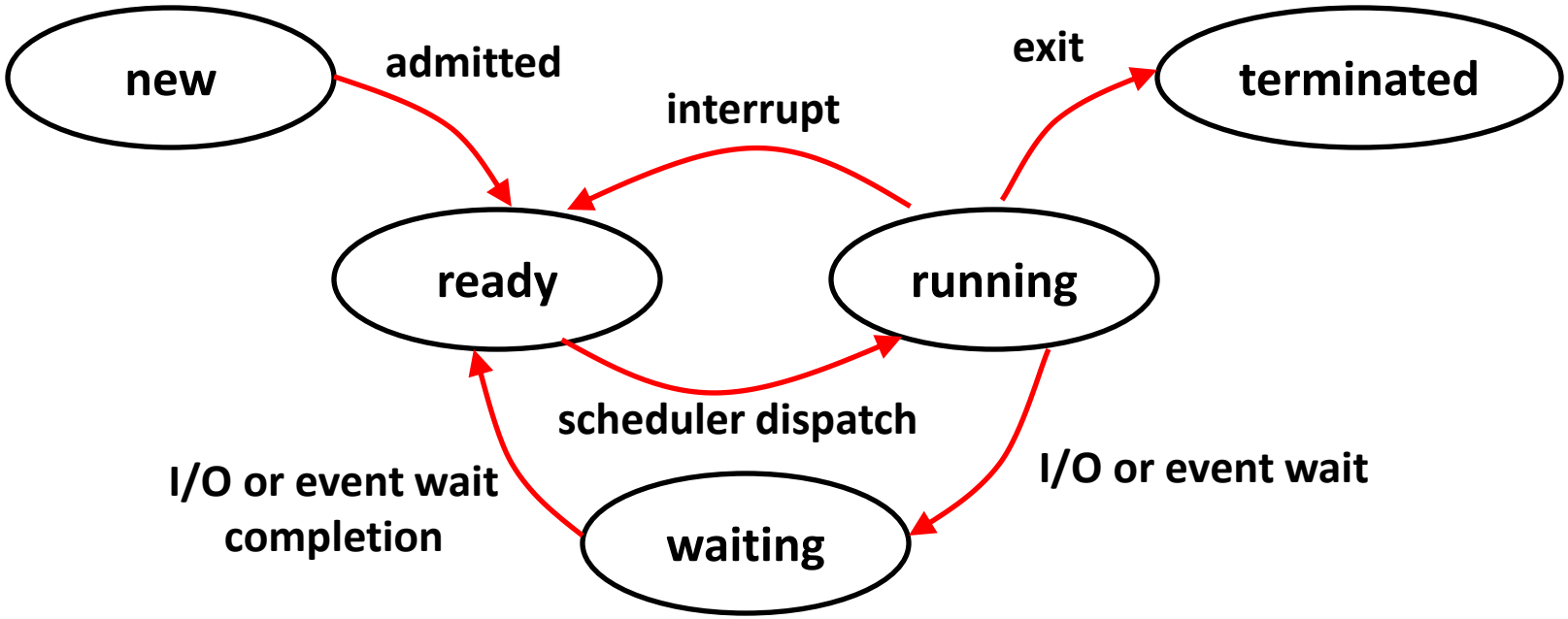
Function calls (nested calls, recursive calls)

Dynamic allocation (object, list, dictionary, tree, etc.)

Global variables (fixed size)

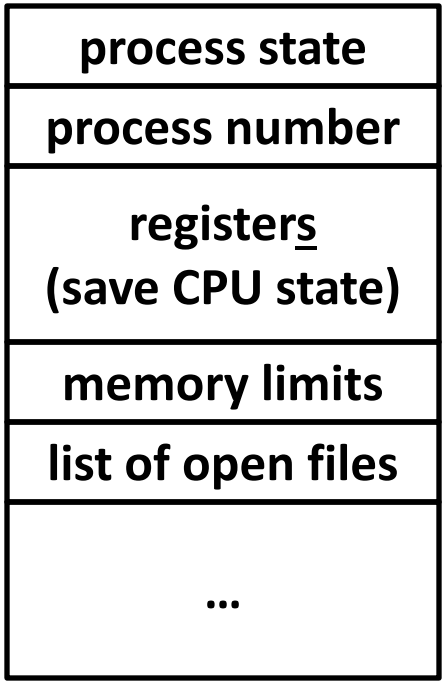
Code (fixed size)

Process state



Process control block

In brief, the PCB simply serves as the repository for any information that may vary from process to process (PCB is a data structure).



PCB

- **Process state**
- **Program counter**
- **CPU registers (PC และ reg อื่น ๆ ทั้งหมด)**
- **CPU-scheduling information**
- **Memory-management information**
- **Accounting information**
- **I/O status information**

Scheduling queues

Job queue: all processes in the system. Ready = in memory
Ready queue: processes that are residing in main memory.
Device queue: processes that are waiting for I/O devices (completion).

Once the process is allocated the CPU and is executing, one of several events could occur:

1. **issue an I/O request and then be placed in the device queue**
2. **create a new subprocess and wait for subprocess's termination**
3. **removed forcibly from the CPU, as a result of an interrupt, and then put back in the ready queue.**

Schedulers

Long-term scheduler (Job scheduler), loading a process into memory.

- execute less often
- control the degree of multi-programming (#processes in memory)
- select a good process mix of I/O-bound and CPU-bound processes

Short-term scheduler (CPU scheduler), allocate CPU to one of them.

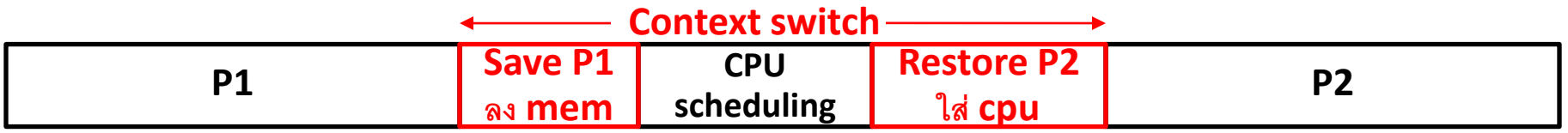
Win + Mac + Linux
มีแค่ CPU scheduler

- execute very often (every 100 milliseconds)
- if the scheduler takes 10 milliseconds to decide to execute a process for 100 milliseconds, then 10/(100 + 10) = 9% overheads

Medium-term scheduler (swapping)

- remove processes from memory (if advantageous)
- when memory requirement overcommitted available memory

Context switch save unloaded process and restore loaded process



Process Creation

Parent / child / pid (คำสั่ง ps และ kill บน Linux)

Concurrent vs. parallel execution

fork() / exec()

wait() – system call

WaitForSignalObject() – Win32 API

```
// compile: gcc fork.c -o fork
// run: ./fork หรือ ./a.out
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main() {

        pid_t pid = fork();

        if (pid < 0) {
                printf("Fork failed");
                return 1;
        } else if (pid == 0) {
                execlp("/bin/ls", "ls", NULL);
        } else {
                wait(NULL);
                printf("Child Complete");
        }
        return 0;
}
```

กิจกรรม **#2 (ทำใน Linux เท่านั้น)**

แก้โปรแกรมนี้ให้สร้างลูก 10 ตัว ลูกแต่ละตัวพิมพ์ข้อความดังนี้ (ดูหน้าถัดไป)

Error! no child

Child

Parent

บน Linux ใช้คำสั่ง man เช่น man fork หรือ man wait เพื่อเรียกดู document ของฟังก์ชันในภาษา C

Parent process (pid = child)

```
if (pid < 0) {
    printf("Fork failed");
    return 1;
} else if (pid == 0) {
    execlp("/bin/ls", "ls", NULL);
} else {
    wait(NULL);
    printf("Child Complete");
}
```

ที่ทำแบบนี้ เพราะเขียนโปรแกรม **OS** ง่าย

fork ก็คือการ **duplicate** หน่วยความจำ

Child process (pid = 0)

```
if (pid < 0) {
    printf("Fork failed");
    return 1;
} else if (pid == 0) {
    execlp("/bin/ls", "ls", NULL);
} else {
    wait(NULL);
    printf("Child Complete");
}
```

Exec คือโหลด **program** ใหม่มาทับ

ตัวอย่าง output ที่ต้องการ

```
I'm the child number 0 (pid 25580)
I'm the child number 1 (pid 25581)
I'm the child number 2 (pid 25582)
I'm the child number 3 (pid 25583)
I'm the child number 4 (pid 25584)
I'm the child number 5 (pid 25585)
I'm the child number 6 (pid 25586)
I'm the child number 7 (pid 25587)
I'm the child number 8 (pid 25588)
I'm the child number 9 (pid 25589)
Parent terminates (pid 25579)
```

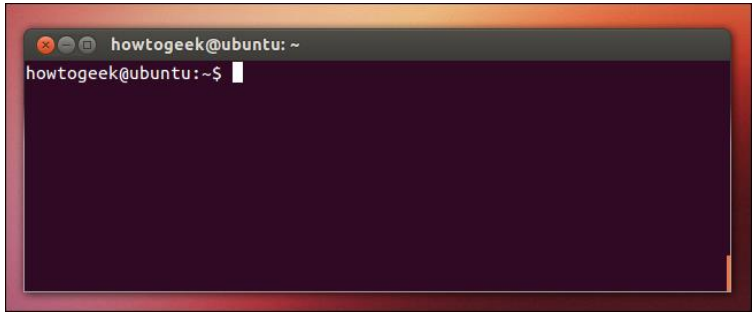
ใช้ฟังก์ชัน getpid()

Parent สร้าง child 10 ตัว
แล้วรอให้ child ทั้ง 10 ตัว terminate ก่อน
จากนั้น parent จึง terminate ตาม

Standard Library (C)

ฟังก์ชัน sleep(1) คือรอ 1 วินาที
ฟังก์ชัน getpid() จะคืนค่า process id
ฟังก์ชัน getppid() จะคืนค่า parent process id

Linux ใช้วิธี fork เพื่อสร้าง process ใหม่ ลอง ps ดู pid ของ bash แล้วเขียนโปรแกรมพิมพ์ pid ของ parent process



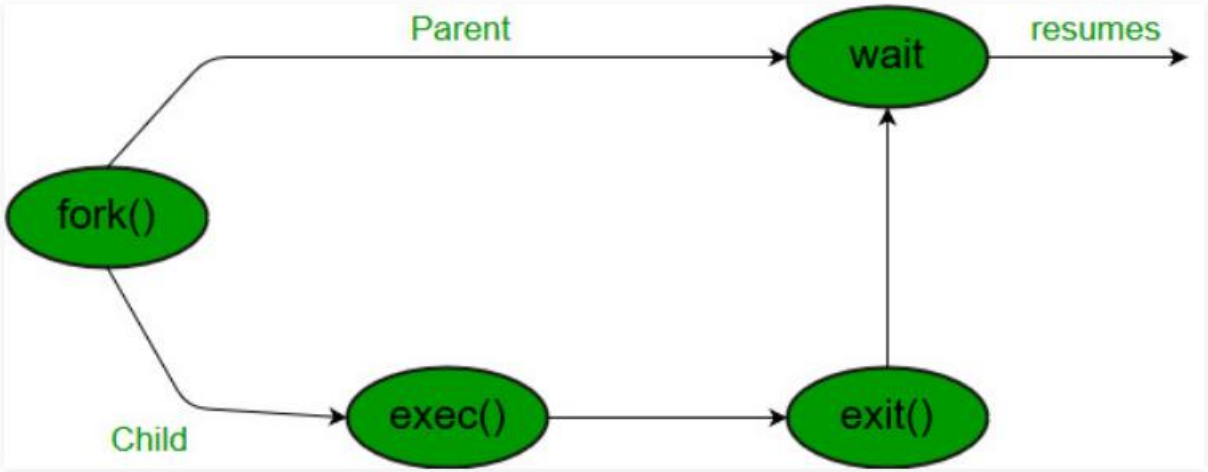
Wait System Call in C

หยุดรอจนกระทั่งลูกตัวใดตัวหนึ่งทำงานเสร็จ
ไม่ใช่รอจนลูกทุกตัวทำงานเสร็จ อ่านเอกสารดี ๆ

Prerequisite : Fork System call

A call to wait() blocks the calling process until one of its child processes exits or a signal is received.
After child process terminates, parent ***continues*** its execution after wait system call instruction.
Child process may terminate due to any of these:

- It calls exit();
- It returns (an int) from main
- It receives a signal (from the OS or another process) whose default action is to terminate.



If there are at least one child processes running when the call to wait() is made, the caller will be blocked until one of its child processes exits. At that moment, the caller resumes its execution. If there is no child process running when the call to wait() is made, then this wait() has no effect at all.

```
// compile: MS Visual C++ 2008 Express

#include "stdafx.h"
#include "windows.h"
int _tmain(int argc, _TCHAR *argv[]) {
        STARTUPINFO si;
        PROCESS_INFORMATION pi;
        ZeroMemory(&si, sizeof(si));
        si.cb = sizeof(si);
        ZeroMemory(&pi, sizeof(pi));
        LPTSTR szCmdline = _tcsdup(TEXT("c:\\windows\\system32\\mspaint.exe"));
        if (!CreateProcess(NULL, szCmdline, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {
                fprintf(stderr, "Create Process Failed");
        }
        WaitForSingleObject(pi.hProcess, INFINITE);
        printf("Child Complete");
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
        return 0;
}
```

ภาษา C ของแต่ละ OS คล้ายกัน
แต่ไม่ได้เหมือนกันทั้งหมด โดยเฉพาะ C Library
เช่น Linux มี fork() แต่ Windows ไม่มี
Windows มี CreateProcess() แต่ Linux ไม่มี

Process Termination

exit() – system call

All the resources of the process – including physical and virtual memory, open files, and I/O buffers – are deallocated by the operating system.

TerminateProcess() – Win32 API

A parent may terminate the execution of one of its children for a variety of reasons, such as these:

1) **the child has exceeded its usage of some of the resources that it has been allocated.**
2) **the task assigned to the child is no longer required.**
3) **the parent is exiting, and the OS does not allow a child to continue if its parent terminates (cascading termination).**

Interprocess communication

There are several reasons for providing an environment that allows process cooperation:

Information sharing	**shared file (lock/unlock)**
Computation speed	**parallel sorting**
Modularity	**web browser (1 web = 1 process)**
Convenience	**copy & paste, drag & drop**

Cooperating processes require Interprocess Communication (IPC).

Shared Memory vs. Message Passing

Shared-memory systems

Producer-consumer problem (สำหรับ users ทั่วไป สอนใช้ pipe บน shell ก็พอ)
เช่น **cat /usr/share/dict/american-english | grep board | wc -l**
เราสามารถเขียนโปรแกรมสร้าง **pipe** เพื่อเชื่อมระหว่าง **2 processes** ได้

Shared memory (queue)

```
#define BUFFER_SIZE 10

typedef struct {
    … … …
} item
// circular queue
item buffer[BUFFER_SIZE];
int in = 0;  // tail
int out = 0; // head
```

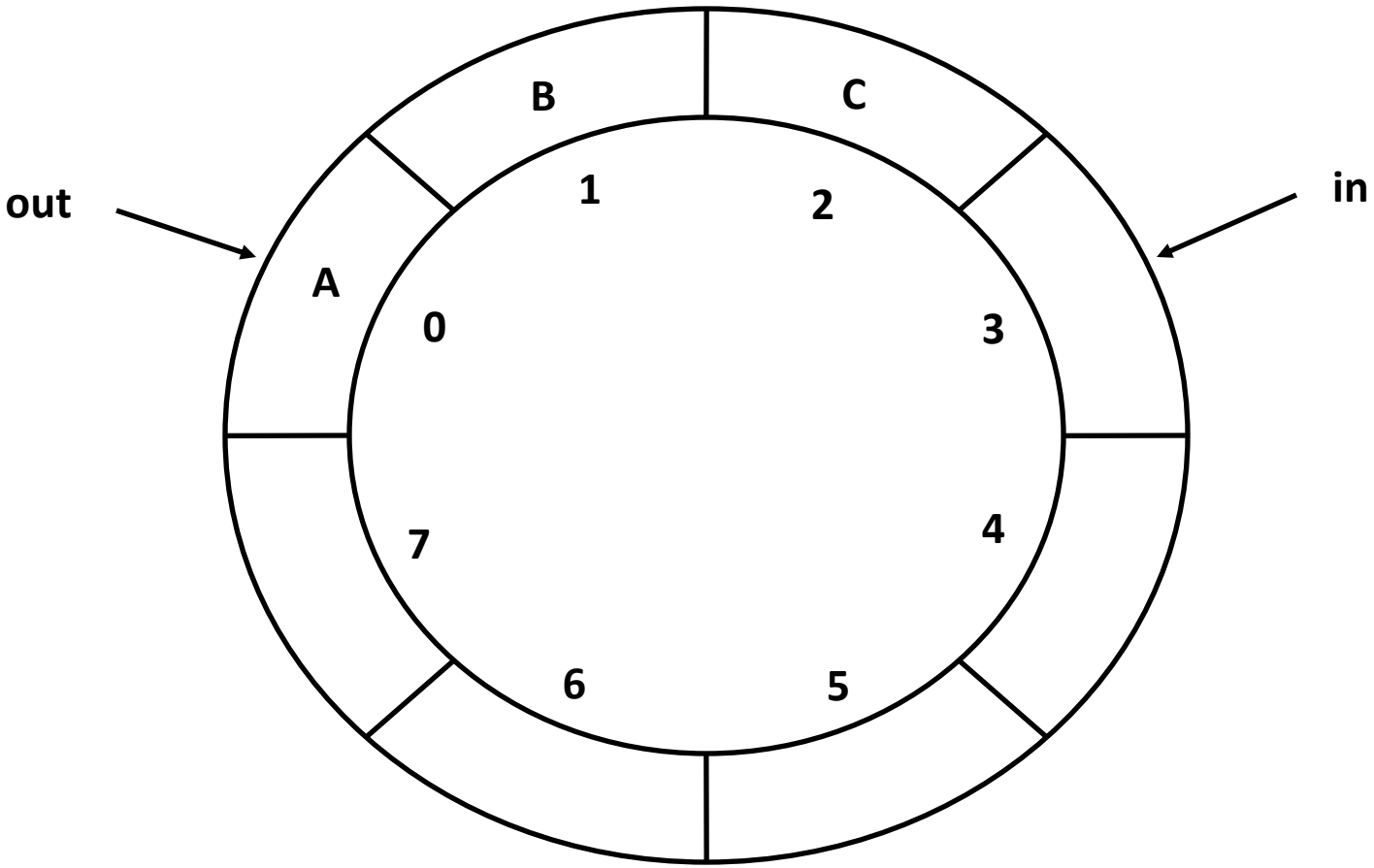
Producer process

```
item nextProduced;
while (ยังต้อง produce อยู่) {
        while (count == BUFFER_SIZE) { }
        buffer[in] = nextProduced;
        in = (in + 1) % BUFFER_SIZE;
        count++;
}
```

Consumer process

```
item nextConsumed;
while (ยังต้อง consume อยู่) {
        while (count == 0) { }
        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        count--;
}
```

Circular Queue



count = 3

BUFFER_SIZE = 8

Pipe and Redirect (one-way communication)

- In C programming, there are 3 stardard I/O channels.
  Standard input (stdin) = keyboard
  Standard output (stdout) = monitor `fprintf(stdout, "%s", "Hello world\n");`
  Standard error (stderr) = monitor `fprintf(stderr, "%s", "Error message\n");`
  Pipe is used to connect stdout (one process) to stdin (another)
  For example, "ls | more" or "tar -c myfolder | gzip -c > myfolder.tar.gz"
- Redirect is used to redirect input or output
  For example, "a.out > file.txt" or "a.out >> file.txt" (append).
  ถ้าใส่ตัวเลข เช่น 1> หรือ 2> หมายถึง กรองเอาเฉพาะ stdout และ stderr
  2>&1 หมายถึง ให้เอา stderr ไปออกที่ stdout แทน
- < gets input from file
  For example, "tar -xvz < filename.tar.gz"

```
// compile: gcc shared_mem.c

#include <sys/types.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <stdio.h>
#include <unistd.h>

int main() {

        int segment_id;
        char *shared_memory;

        segment_id = shmget(IPC_PRIVATE, 4096, S_IRUSR | S_IWUSR);
        shared_memory = (char *)shmat(segment_id, NULL, 0);

        sprintf(shared_memory, "Hi there!");

        // continue on the next page
```

```
        pid_t pid;

        pid = fork();

        if (pid == 0) {                                          // child

                printf("%s\n", shared_memory);

        } else {                                                 // parent

                wait(NULL);
                shmdt(shared_memory);                            // detach
                shmctl(segment_id, IPC_RMID, NULL);              // remove
        }

        return 0;
}
```

Client-Server Systems

Sockets, ports

sock.accept() is a blocking method.

127.0.0.1 is loop back.

```
import java.net.*;
import java.io.*;

public class DateServer
{
   public static void main(String[] args) {
      try {
         ServerSocket sock = new ServerSocket(6013);

         // now listen for connections
         while (true) {
            Socket client = sock.accept();

            PrintWriter pout = new
             PrintWriter(client.getOutputStream(), true);

            // write the Date to the socket
            pout.println(new java.util.Date().toString());

            // close the socket and resume
            // listening for connections
            client.close();
         }
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

Figure 3.19 Date server.

กิจกรรม **#3 (ทำใน Linux เท่านั้น)**
ต้อง new process หรือ spawn thread
เพื่อให้บริการ client ได้หลาย ๆ ตัวพร้อมกัน

วน loop ส่งข้อมูลวันเวลาปัจจุบันไปเรื่อย ๆ

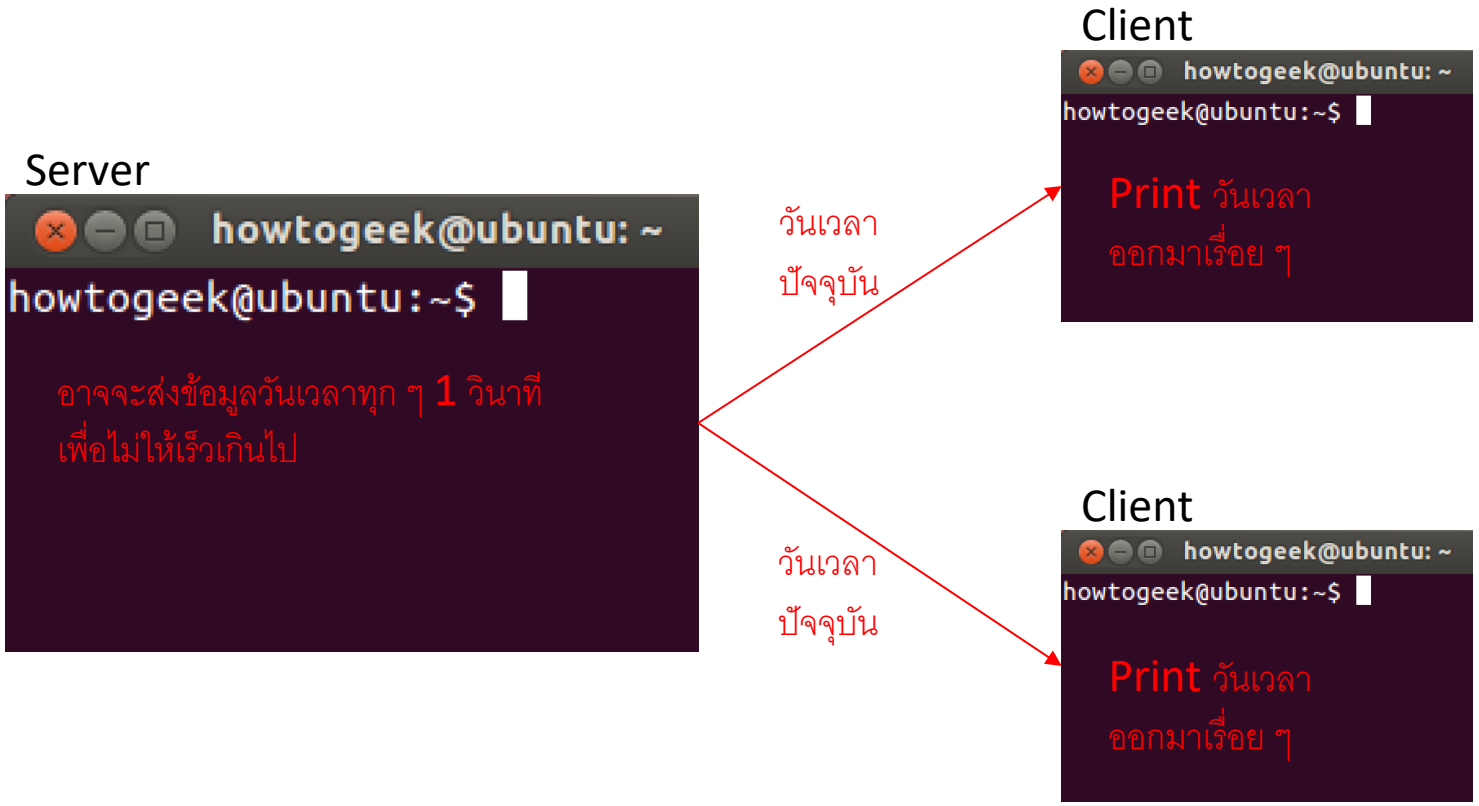
ใช้ภาษาโปรแกรมอื่น ๆ ก็ได้ ที่สร้าง kernel thread ได้

```
import java.net.*;
import java.io.*;

public class DateClient
{
   public static void main(String[] args) {
      try {
         //make connection to server socket
         Socket sock = new Socket("127.0.0.1",6013);

         InputStream in = sock.getInputStream();
         BufferedReader bin = new
            BufferedReader(new InputStreamReader(in));

         // read the date from the socket
         String line;
         while ( (line = bin.readLine()) != null)
            System.out.println(line);

         // close the socket connection
         sock.close();
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

Loop back และ port number

Figure 3.20 Date client.

ใช้ภาษาโปรแกรมอื่น ๆ นอกจาก **Java** ก็ได้



ต้องเขียนโปรแกรมให้ fork (สร้าง process/thread ใหม่) เพื่อบริการ client แต่ละตัว
ถึงจะเห็นว่า client ทุกตัวได้รับบริการไปพร้อม ๆ กัน

Message Passing

MP is useful in distributed environment, where the communicating processes may reside on different computers connected by a network (e.g. Cluster).

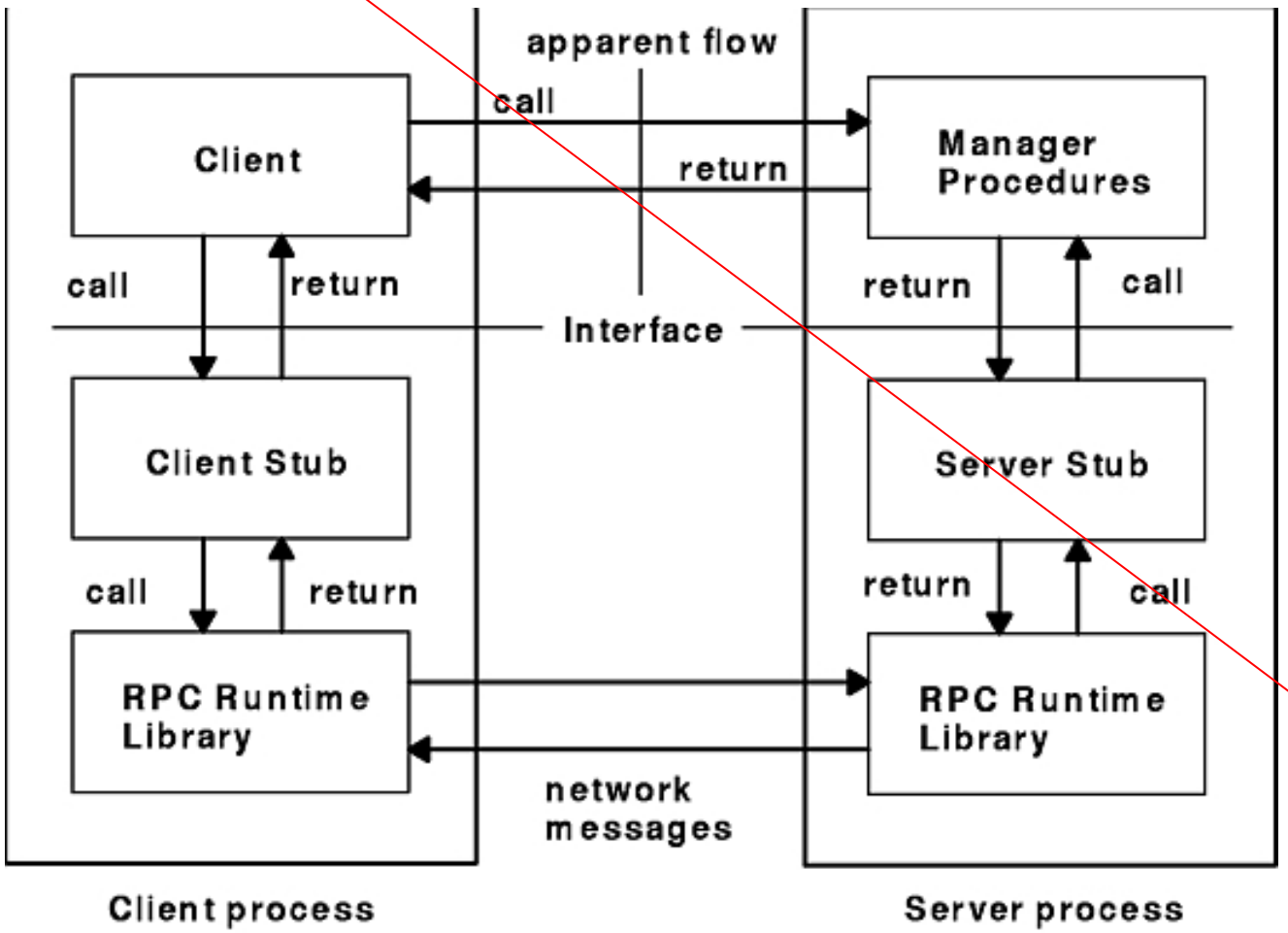
OS สร้าง message queue ทำหน้าที่คล้าย ๆ กับ mail box

แบบ 1. หลาย ๆ process บนเครื่องเดียวกัน มี mail box ชุดเดียว

แบบ 2. มีหลาย ๆ เครื่อง แต่ละเครื่องต้องมี mail box ของตัวเอง

Remote Procedure Calls

คล้ายๆ virtualization หลอกว่าฟังก์ชันนั้น execute อยู่บนเครื่องจริงๆ



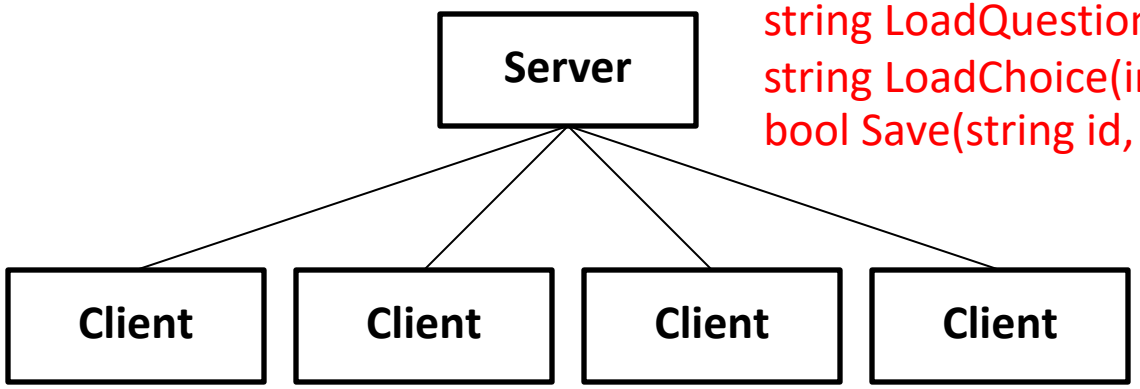
Remote Procedure Call Flow

Programmer's view

What actually happen

Why RPC

- If no RPC, you have to use socket programming and you have to impose a structure on data! (Java uses serialization)
- Only one copy of a function, updating a service at one place, i.e. banking application.
- Distributing jobs to many servers. Each server is good at a particular job. Parallel programming!
- Easy interprocess communication (no socket programming, no imposing data), i.e. e-exam (multiple choices).

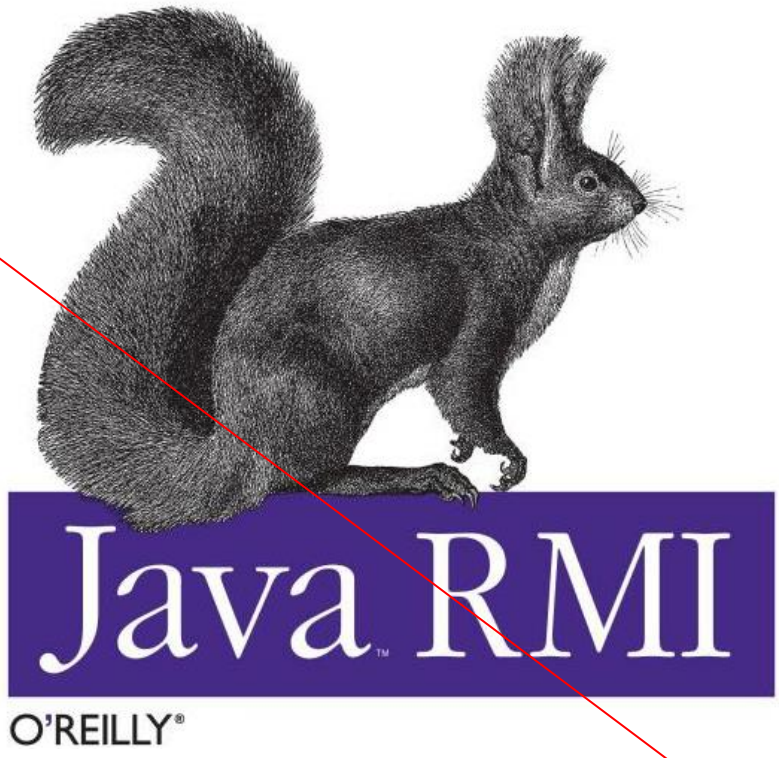


string LoadQuestion(int i)
string LoadChoice(int i, int j)
bool Save(string id, int i, int j)

Faster coding compared to socket programming or web programming.

Today's RPC

- .NET Remoting
- Java RMI (remote method invocation)



ข้อเสียคือทั้ง **server** และ **client** ต้องเป็น **platform** เดียวกัน

Web Services

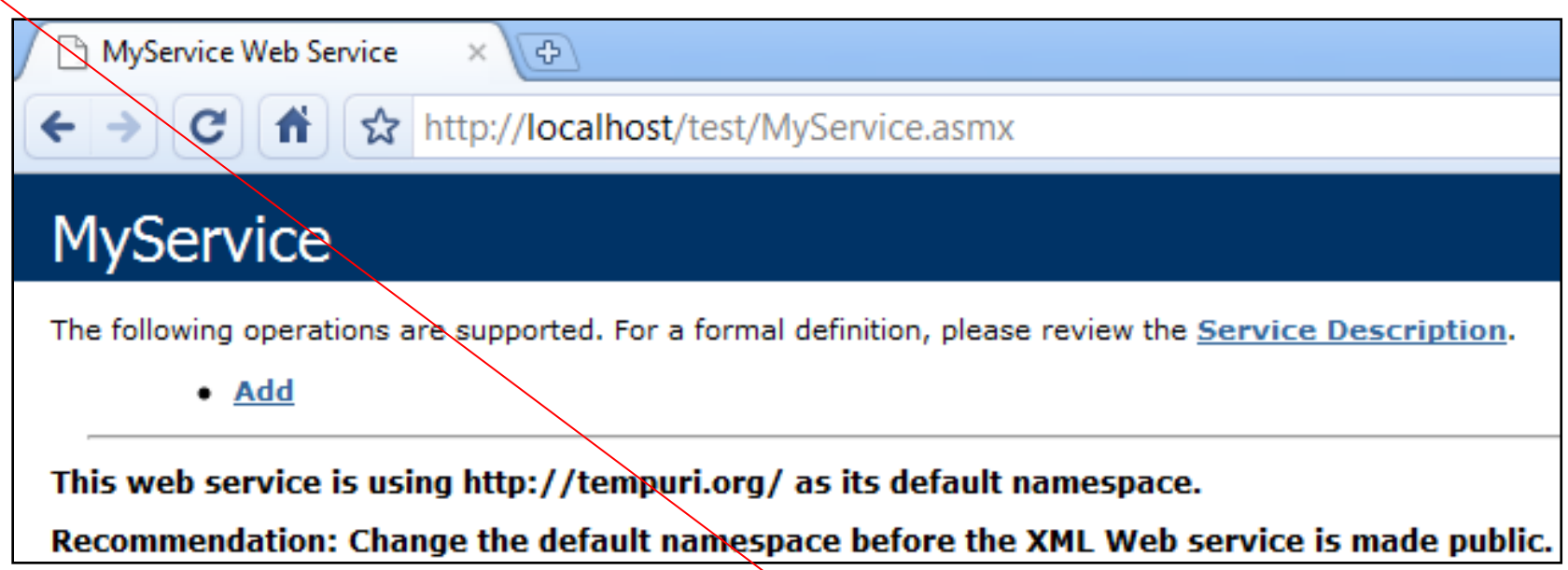
- ใช้ภาษาอื่น ๆ นอกจาก C# ก็เขียน web service ได้
- เรียกใช้ web service จากภาษาใดก็ได้
- แก้ปัญหาซอฟต์แวร์เถื่อน คิดค่าบริการจากจำนวนครั้งที่ใช้งาน หรือระยะเวลาที่ใช้งานได้

MyService.asmx
(in a virtual directory)

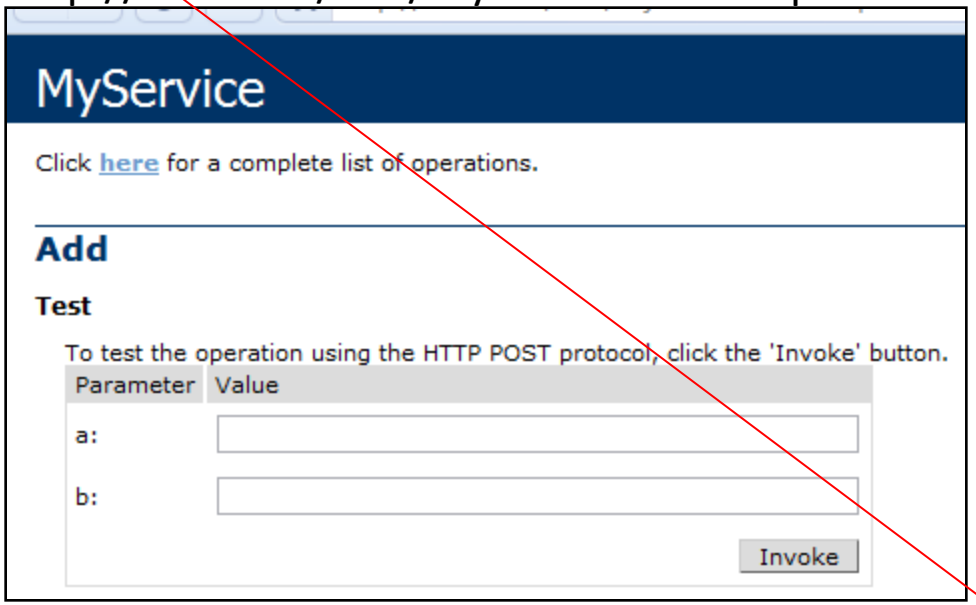
```
<%@WebService Language="C#"
Class="MyService"%>

using System.Web.Services;

public class MyService : WebService {

    [WebMethod]
    public int Add(int a, int b) {

        return a + b;
    }
}
```

Client
(MyService.dll)

```
using System;
using MyNameSpace;

class MyProg {

    public static void Main(string [] Args) {

        MyService S = new MyService();
        Console.WriteLine(S.Add(1,2));
    }
}
```



http://localhost/test/MyService.asmx?op=Add



Process Communication

ปัจจุบันเรามักจะต้องเขียนโปรแกรมของเราให้ communicate กับโปรแกรมสำเร็จรูปอื่นๆ (ซึ่งอาจจะอยู่บนเครื่องเดียวกันหรือคนละเครื่องก็ได้)

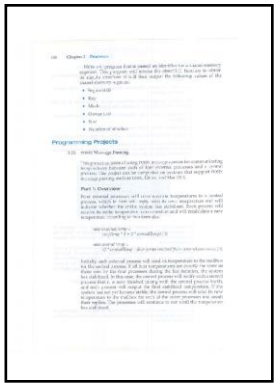
.NET (C#) simplified code

```
ExcelApp = new Excel.ApplicationClass();   สร้าง process รัน Excel
ExcelBook = ExcelApp.Workbooks.Open(Filename, …, …, …);
ExcelSheet = ExcelBook.Worksheets.get_Item(1);

// write
 ExcelSheet.Cells[row,col]  = “Chatchawit”;

// read
 name = ExcelSheet.Cells[row,col];

ExcelBook.Close(true, Missing.Value, Missing.Value);
ExcelApp.Quit();
```

ปัจจุบันสามารถอ่านเขียนไฟล์ **Excel** ได้ตรง ๆ เลย

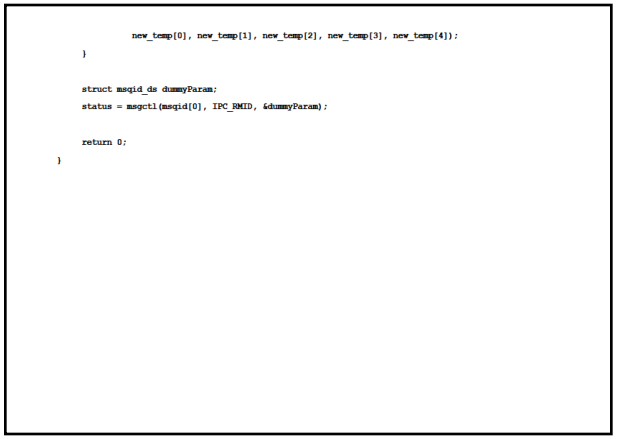
Homework

โจทย์



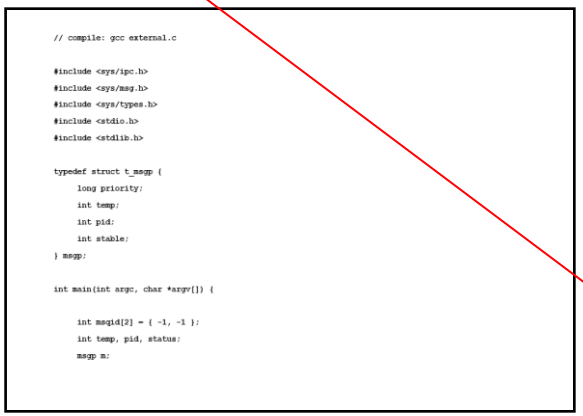
POSIX Message Passing (p. 148)

- write “central.c” and “external.c”
- 1 copy of “central” and 4 copies of “external” can be started in any order.
- begin from 2 processes and one-way communication.
- then, two-way communication (2 mailboxes) and 5 processes (5 mailboxes).
- use 5 terminals for the ease of debugging.
- always check and clear message queues by “ipcs” and “iprm -q qid.”



```
            new_temp[0], new_temp[1], new_temp[2], new_temp[3], new_temp[4]);
    }

    struct msqid_ds dummyParam;
    status = msgctl(msqid[0], IPC_RMID, &dummyParam);

    return 0;
}
```

central.c



```
// compile: gcc external.c

#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct t_msgp {
    long priority;
    int temp;
    int pid;
    int stable;
} msgp;

int main(int argc, char *argv[]) {

    int msqid[2] = { -1, -1 };
    int temp, pid, status;
    msgp m;
```

external.c

Central

```
// compile: gcc central.c

#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct t_msgp {
     long priority;
     int temp;
     int pid;
     int stable;
} msgp;

int main(int argc, char *argv[]) {

     int temp[5];
     int new_temp[5];
     int msqid[5] = { -1, -1, -1, -1, -1 };
     int status, stable, i;
```

บังคับว่าตัวแรกใน struct ต้องเป็น message type (หรือจะมองว่าเป็น priority ก็ได้)

size ของ message ที่จะส่งในฟังก์ชัน msgrcv นับแค่นี้

```
msgp m;

temp[0] = atoi(argv[1]);

msqid[0] = msgget(70, 0600 | IPC_CREAT);
while (msqid[1] < 0) msqid[1] = msgget(71, 0600);
while (msqid[2] < 0) msqid[2] = msgget(72, 0600);
while (msqid[3] < 0) msqid[3] = msgget(73, 0600);
while (msqid[4] < 0) msqid[4] = msgget(74, 0600);

stable = 0;

while (!stable) {

    // receive 4 messages (unsorted!)
    for (i = 1; i <= 4; i++) {
        status = msgrcv(msqid[0], &m, sizeof(msgp) - sizeof(long), 2, 0);
        temp[m.pid] = m.temp;
    }

    // update temp
```

แปลง string เป็น int

อุณหภูมิเริ่มต้น

msqid คือ message queue id, 70 คือ queue name (users 2 คนอาจจะใช้ queue name ซ้ำกันได้)

0600 คือยอมให้ process ของผู้สร้างอ่านเขียน mailbox ได้

IPC_CREAT คือ ถ้ายังไม่มีให้สร้างใหม่

ขอ message queue id ของ external 4 ตัว

while loop เพื่อรอให้ external สร้าง

0 อ่าน first message บน queue

>0 เช่น 2 เอา first message ที่ priority = 2

<0 เช่น -2 เอา first message ที่ priority <= 2

blocking

```
new_temp[0] = (temp[0] + temp[0] + temp[1] + temp[2] + temp[3] + temp[4]) / 6;
for (i = 1; i <= 4; i++) {
     new_temp[i] = ((3 * temp[i]) + (2 * temp[0])) / 5;
}

// check stability
if (temp[1] == new_temp[1] && temp[2] == new_temp[2] &&
    temp[3] == new_temp[3] && temp[4] == new_temp[4]) {
     stable = 1;
}

// send 4 messages (sorted!)
for (i = 1; i <= 4; i++) {
     m.priority = 2;
     m.temp = new_temp[i];
     m.pid = i;
     m.stable = stable;
     status = msgsnd(msqid[i], &m, sizeof(msgp) - sizeof(long), 0);
}

printf("%6d %6d %6d %6d %6d\n",
```

update central

update external

blocking

1 central + 4 external

```
            new_temp[0], new_temp[1], new_temp[2], new_temp[3], new_temp[4]);
    }

    struct msqid_ds dummyParam;
    status = msgctl(msqid[0], IPC_RMID, &dummyParam);

    return 0;
}
```

remove ใช้กรณี IPC_STAT IPC_SET