Excercine 3

1. LINE A = 0 เนื่องจาก ค่า ที่ process parent เป็น 0
2. 8 Process มีการ fork 3 ครั้ง ซึ้งเท่ากับ 2^3

3 เมื่อมีการสร้าง Process ใหม่ ด้วยคำสั่ง fork Process หลัก และ Child Process จะแบ่งเซ็กเม้นกันนั้นคือสถาณะ Shared Memory Segment

4. **Short-term (CPU scheduler)** เลือกว่าจะเอางานไหนมารัน ใช้เวลาสั้นกว่า ทำให้ทำงานถี่กว่า

**Medium-term scheduling** เป็นตัวทำงาน swap in, swap out เพื่อช่วยในส่วนของการ balance ระหว่าง process สองชนิด มีประโยชน์ในการลบ Process ออกจากหน่วยความจำและลดระดับ multiporgramming

**Long-term (job scheduler)** เลือกว่าจะเอางานไหนจากใน pool มาใส่ใน ready เช่น Grid Engine

รันไม่ถี่มาก ใช้เวลานานกว่า และ เป็นตัวควบคุมการรันโปรแกรม

5. ระบบปฏิบัติการเรียกใช้ตัว scheduler เพื่อตัดสินใจว่า process ไหนต้องดำเนินการต่อ

6. ทำหน้าที่ในการอ่านการตั้งค่า เพื่อดูว่าจะใช้ รัน Level ไหน นอกจากนั้นยังทำหน้าที่

จัดการว่าใครต้องรันตอนไหนโดยอ่านค่าที่กำหนดไว้และมันยังตามมาดูแลโปรเซสให้หลังจากกระบวนการบูท เช่น สามรถดูการทำงานของ service ได้ผ่าน system

7. **Zombie Process** คือ process ที่ไม่ได้ดำเนินการต่อไป แต่ยังมีหมายเลข PID อยู่ process นี้จะไม่หายไป

จนกว่า process parent จะปิดและรีสตาร์ทระบบ สาเหตุที่ทำให้เกิด zombie process คือ parent process

ยังไม่ได้รับ system call จาก child process จึงทำให้เกิด zombie process

**orphan process** คือ process ที่ parent process เสร็จสิ้นหรือสิ้นสุดแม้ว่ามันจะยังทำงานอยู่

ตัวอย่าง เคสที่ Parent ทำการ terminate ไปแล้ว ในขณะที่ Child ยัง execute ไม่เสร็จ และไม่เกิดการ cascading (OS อนุญาติให้ child มีชีวิตต่อ) กรณีนี้ เรียก child ว่า orphan

8. code C++ เพื่อคำนวณ Collatz Conjecture with Shared-Memory Object

#include <stdlib.h>

#include <stdio.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main(){

int n;

printf("\nEnter a positive number: ");

scanf("%d", &n);

if(n>0){

collatz(n);

}

else{

printf("please enter a positive number.\n");

}

return 0;

}

int collatz(int i){

pid\_t child\_pid, wpid;

int waiting = 0;

if ((child\_pid = fork())==0){

while(i != 1){

printf("%d ", i);

if(i%2==0){

i/=2;

}

else{

i = 3\*i+1;

}

}

printf("1\n");

exit(1);

}

while((wpid = wait(&waiting))>0){

printf("child end\n");

}

return;

}

}

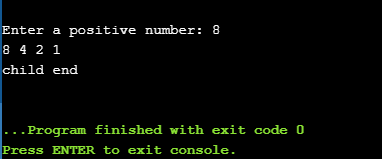
Output ที่รัน เมื่อใส่เลขคี่15

Output Enter a positive number: 15

15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1

child end

output เมื่อใส่เลขคู่ 8



9. A. messeage-passing model

B. Mon May 06 04:49:56 ICT 2019

C. ตัวรับการสื่อสารระหว่างโปรแกรมบนเครื่อข่ายเพื่อแสดงให้การคุยกันระหว่างเครือข่ายแต่ละตัว

D. ถ้ารัน DateClient ก่อน DateServer จะขึ้น error ว่า java.net.ConnectException: Connection refused: connect

Ex4

1. -User-level threads จะเป็นแบบ Many-to-one หมายถึง ระบบปฏิบัติการจะวาง Thread ทั้งหมดของ Multithread process ไว้ใน Execution context เพียงตัวเดียว ทำงานอยู่ภายใต้พื้นที่ของ User ไม่เกี่ยวข้องกับพื้นที่ของ Kernel จึงทำให้ไม่สามารถเข้าใช้งาน Kernel ได้โดยตรง

Kernel-level threads จะเป็นแบบ One-to-one หมายถึง ระบบปฏิบัติการจะวาง Thread แต่ละตัวไว้ใน Execution context ของตัวเอง ดังนั้น ในการวาง Thread แต่ละตัวจึงต้องให้ระบบปฏิบัติการช่วยในเรื่องการ Mapping ตัว User-level threads แต่ละตัวเข้ากับ Kernel-level threads

-เมื่อการทำงานแบบ Thread เดียว เพื่อความรวดเร็วกว่า Thread ในระดับ kernel และ kernel จะไม่มีส่วนร่วมในการ synchronize

-เมื่อระบบปฏิบัติการบล็อกการร้องขอใช้งานอินพุต/เอ้าท์พุต Thread ตัวใดแล้วจะไม่ส่งผลให้ Thread ตัวอื่นทั้งหมดถูกบล็อกไปด้วย แต่จะตัวที่พร้อมจะประมวลผลจะถูกเลือกให้มาประมวลผลแทน

2. บริการของเว็บที่ให้บริการคำขอรายการในชุดข้อความที่แยกต่างหากหรือโปรแกรมเชิงโต้ตอบซึ่ง Thread ใช้เพื่อตรวจสอบ อินพุตของผู้ใช้ Thread อื่นแสดง application ที่รันอยู่และ Thread ที่ 3 จะใช้เพื่อตรวจสอบประสิธิภาพ ด้วยตัวอย่างที่ยกมาทำให้ Multiple Kernel Thread มีประสิธิภาพมากกว่า แบบ Single Thread

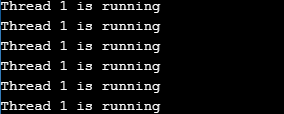
3. Heap Memory กับ Global Variables จะ แชร์กันใน MultiThread Process

4. 6Process   
 2Thread

5. Thread Pool เป็น Class ที่ .NET Framework เตรียมเอาไว้สำหรับการทำงานแบบ Multi-thread ที่ไม่ซับซ้อน Thread Pool นั้น Run แบบ Background Thread ดังนั้น Application สามารถหยุดการทำงานได้ทันทีถ้า Foreground Thread อื่นๆ สิ้นสุดการทำงานลงThread Pool เหมาะสำหรับที่จะใช้จัดการ Thread ที่ทำงานเล็กๆ ไม่ยุ่งยากซับซ้อนและใช้เวลาในการประมวลผลไม่น้อย รวมทั้งไม่คำนึงถึงลำดับความสำคัญของ Thread

6. class ThreadTest extends Thread {   
public void run() {   
 try{   
//แสดง Thread ที่กำลังทำงานอยู่  
System.out.println ("Thread " +   
 Thread.currentThread().getId() + " is running");   
 }   
 catch (Exception e)   
{   
//Throwing an exception  
 System.out.println ("Exception is caught");   
 }   
 }   
}   
 public class Main {   
public static void main(String[] args) {   
 int n = 8;   
 for (int i=0; i<n; i++) {   
ThreadTest object = new ThreadTest();   
//start() แทนที่ด้วย run() เพื่อเริ่มต้น  
 object.run();

}}}



CodeOpenMP

#include <omp.h>

#include <stdio.h>

int main(int argc, char \*argv[])  
{  
 /\* sequential code \*/  
 #pragma omp parallel //สร้าง thread ให้มากที่สุดเท่าที่ cores มี  
 {  
 printf("I am a parallel.");  
}  
/\* sequential code \*/

return 0;

}  
Output 

7. output ของ child = 5 หลังจาก fork แล้ว parent process จะรอให้ child process เสร็จ Thread ใหม่จะถูกสร้างขึ้นสำหรับ child process และเรียกใช้ฟังก์ชั่น ซึ้งค่า = 5 LINE C = 5

หลังจากเสร็จสิ้น process child ค่าของ global variable จะอยู่ใน parent process จะ เท่ากับ 0

LINE P = 0