

CS222

Operating Systems

Lecture 06

Process Concept: Practice

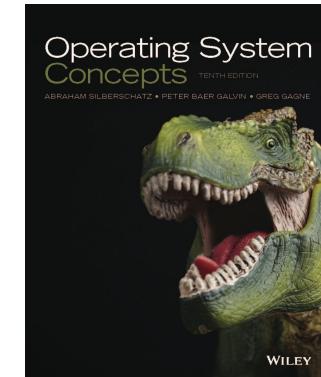
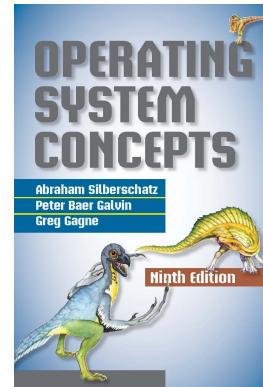
(Section 100001)

ผศ. ดร. กษิิดิศ ชาญเชี่ยว

ckasidit@tu.ac.th

Textbook

- Avi Silberschatz, Peter B. Galvin and Greg Gagne; Operating System Concepts, 9th Edition; John Wiley & Sons, Inc; 2012; ISBN 978-1118063330



- Original Slides
- <https://www.os-book.com/OS9/slide-dir/index.html>
- <https://www.os-book.com/OS10/slide-dir/index.html>

Textbook

- Advanced Programming in the UNIX Environments
- <http://www.apuebook.com/>

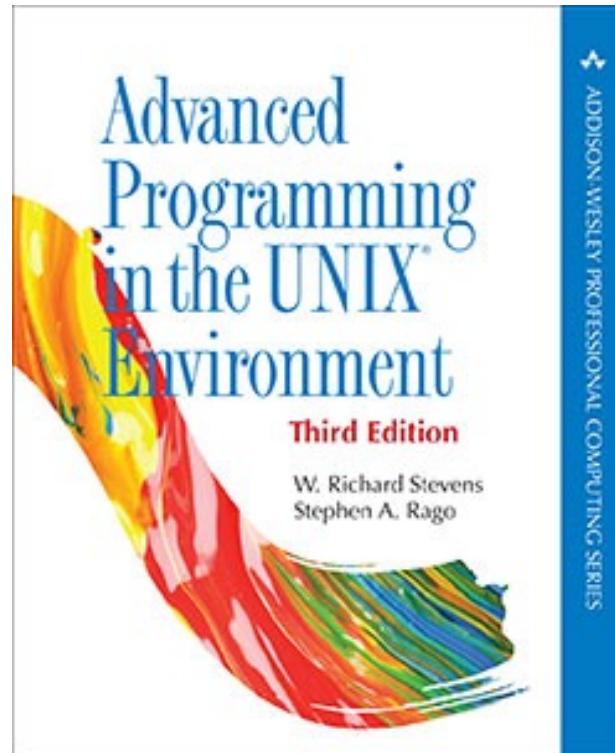
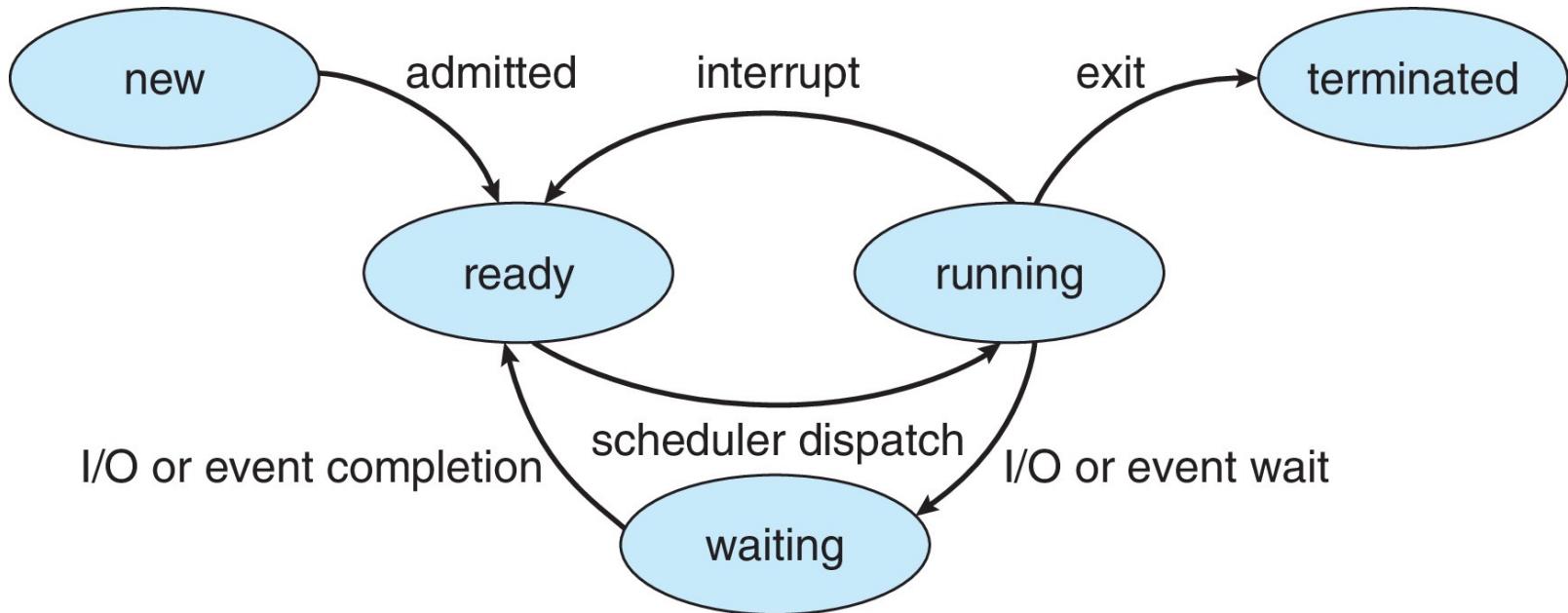




Diagram of Process State





Linux Process State

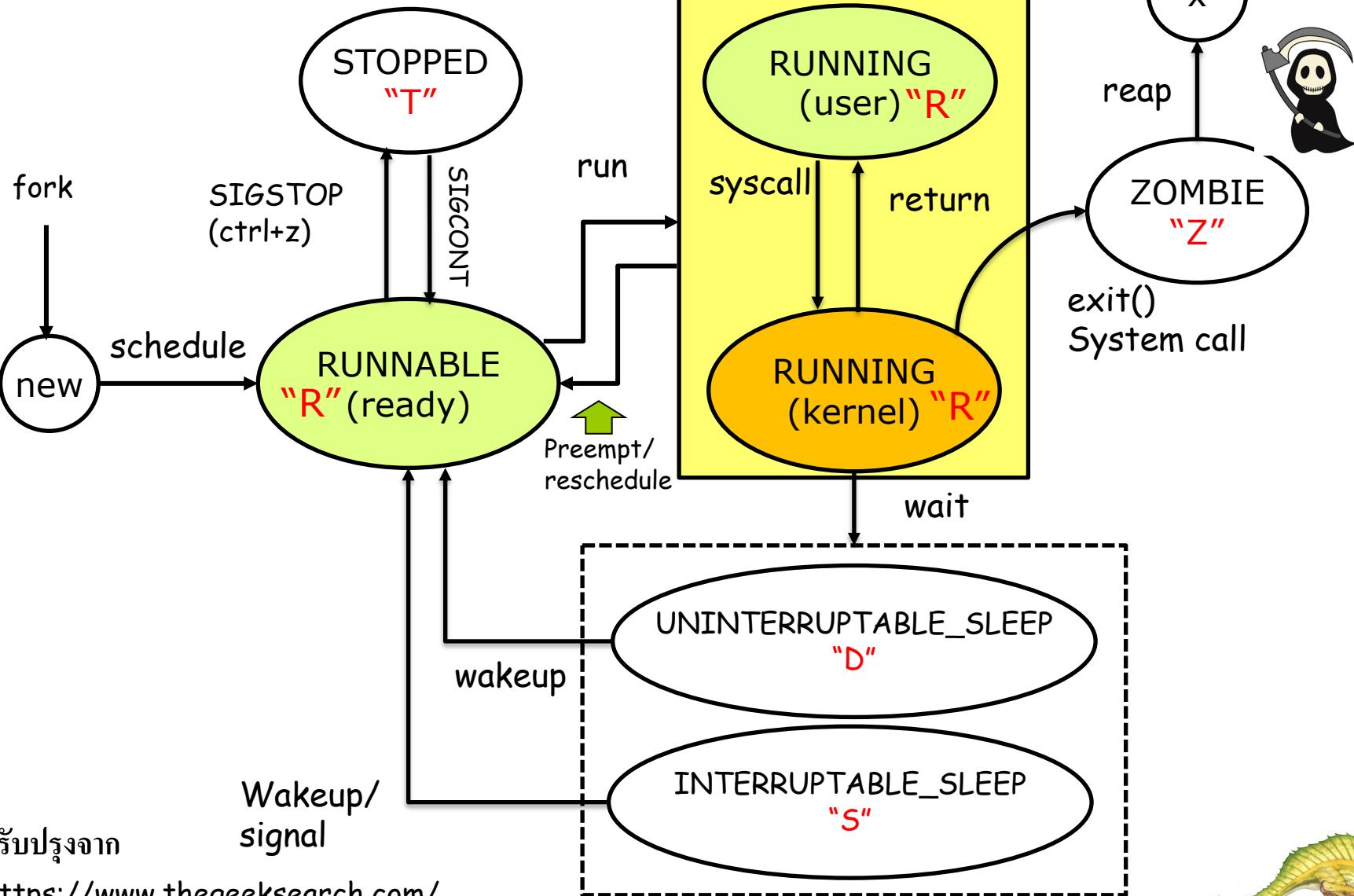
- หาได้จากคำสั่ง “top” หรือ “ps aux”
- มี 5 สถานะได้แก่
 - 'D' = UNINTERRUPTABLE_SLEEP
 - 'R' = RUNNING & RUNNABLE
 - 'S' = INTERRUPTABLE_SLEEP
 - 'T' = STOPPED
 - 'Z' = ZOMBIE

<https://www.cbtnuggets.com/blog/certifications/open-source/what-are-the-5-linux-process-states>





Linux Process State



ปรับปรุงจาก

[https://www.thegeeksearch.com/
beginners-guide-to-managing-processes-running-on-a-linux-system/](https://www.thegeeksearch.com/beginners-guide-to-managing-processes-running-on-a-linux-system/)





สถานะ

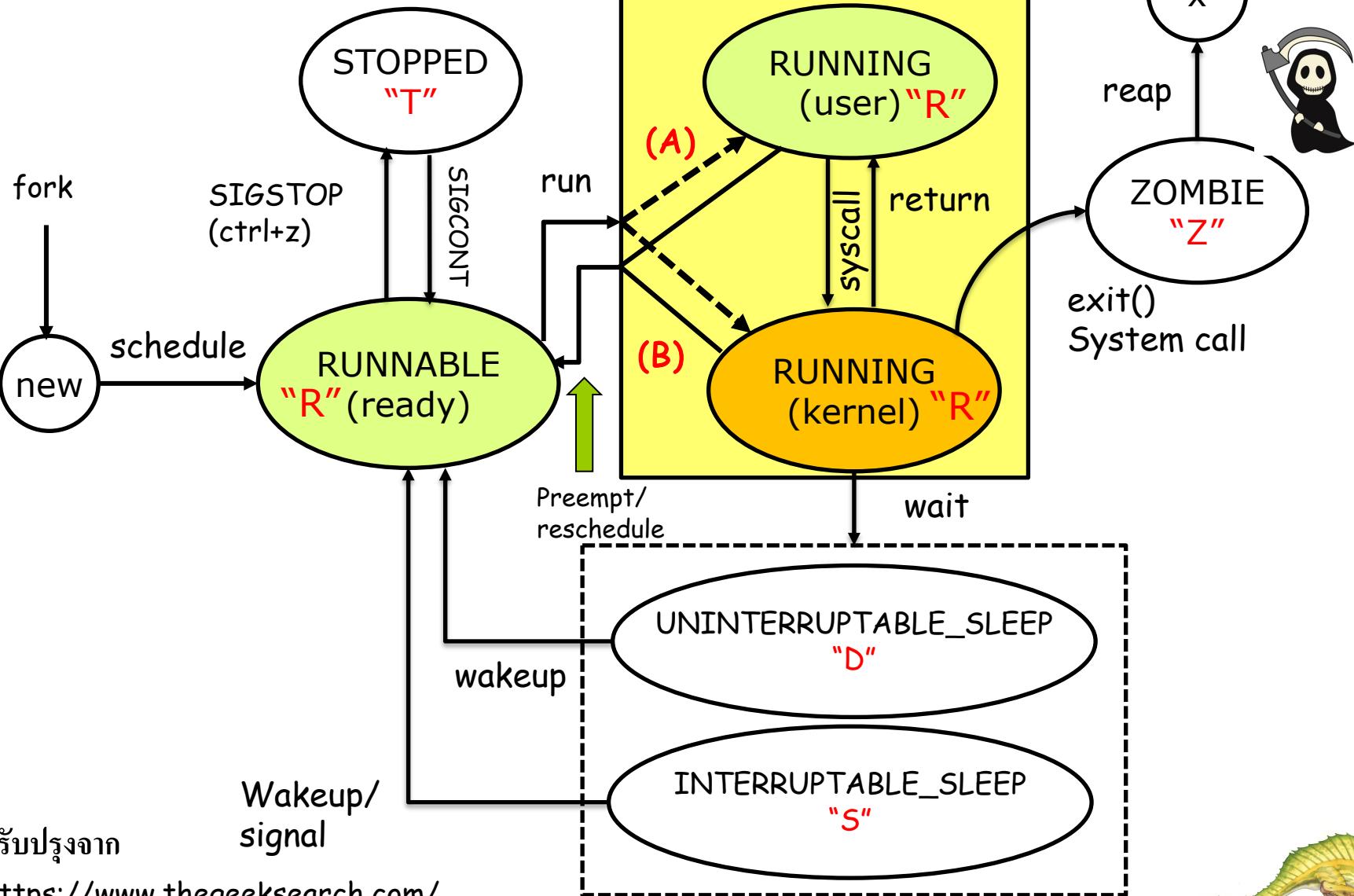
- 'R' = RUNNING & RUNNABLE ประกอบไปด้วยสถานะสองแบบใหญ่ๆ คือ RUNNABLE และ RUNNING
 - สถานะ RUNNABLE เป็นสถานะของ Process ที่พร้อมจะถูกรันบนซีพียู ซึ่งจะรอให้ Process Scheduler เลือกให้ซีพียูประมวลผล
 - สถานะ RUNNING เป็นสถานะของ Process ที่กำลังรันบนซีพียู Process อาจรันใน user mode เมื่อรัน Application Program และรันใน kernel mode เมื่อ Application Program เรียกใช้ system call (syscall) เมื่อ system call เสร็จก็จะ return กลับไปประมวลผลใน user mode ตามเดิม

<https://www.cbt nuggets.com/blog/certifications/open-source/what-are-the-5-linux-process-states>





Linux Process State



ปรับปรุงจาก

[https://www.thegeeksearch.com/
beginners-guide-to-managing-processes-running-on-a-linux-system/](https://www.thegeeksearch.com/beginners-guide-to-managing-processes-running-on-a-linux-system/)





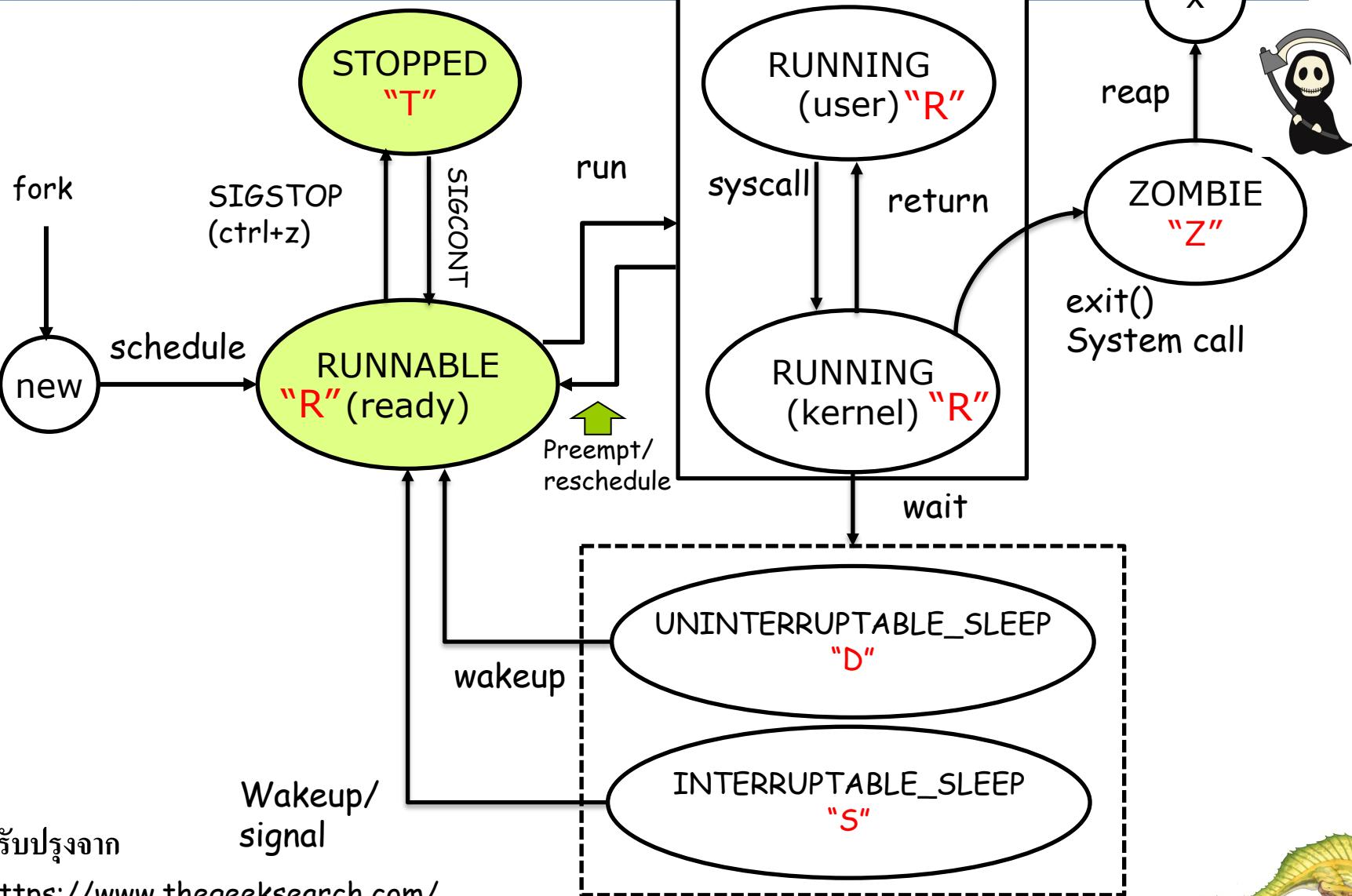
สถานะ

- เมื่อ Process เปลี่ยน State จาก RUNNABLE เป็น RUNNING สถานะ RUNNING อาจเป็น การรันใน user mode หรือ kernel mode ก็ได้
- (A) ในการณ์ที่ Process รันเป็นครั้งแรก หรือ Process ถูก Timer Interrupt ในรอบที่แล้วขณะประมวลผลใน User mode **มันจะเปลี่ยนจาก RUNNABLE เป็น RUNNING ใน user mode**
- (B) ในสถานการณ์ที่ Process ถูกขัดจังหวะด้วย Timer Interrupt ในขณะที่ มันกำลังประมวลผลใน kernel mode (เช่นอยู่ระหว่างเรียกใช้ system call) Process จะถูกย้ายไปเป็น RUNNABLE state และมีค่า mode เป็น kernel mode **เมื่อมันได้รับ Time Slice อีกครั้ง Scheduler จะเปลี่ยน State จาก RUNNABLE เป็น RUNNING ใน kernel mode**





Linux Process State



ปรับปรุงจาก

[https://www.thegeeksearch.com/
beginners-guide-to-managing-processes-running-on-a-linux-system/](https://www.thegeeksearch.com/beginners-guide-to-managing-processes-running-on-a-linux-system/)





สถานะ

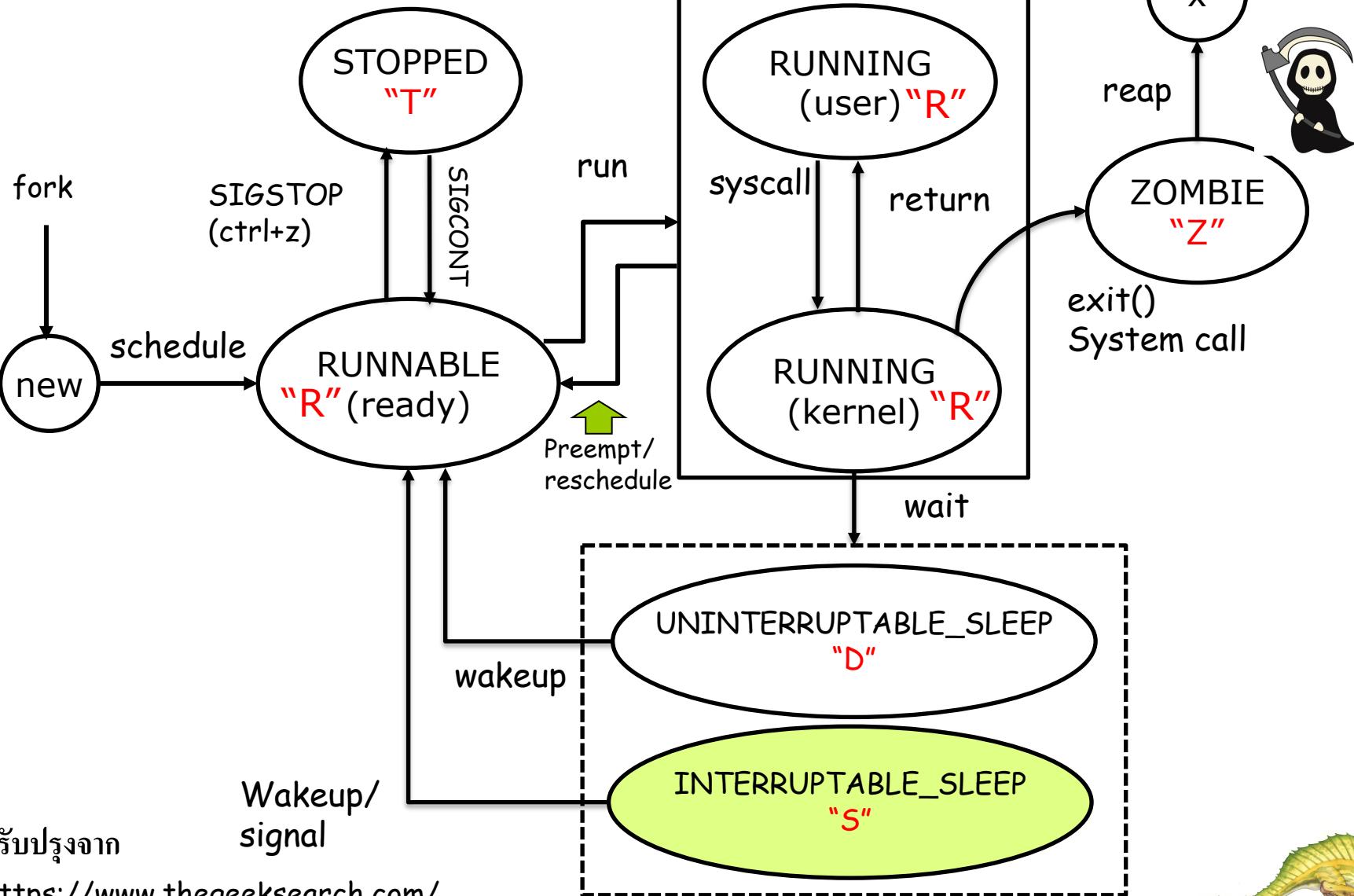
- 'R' = RUNNING & RUNNABLE (ต่อ)
 - ในกรณีที่เกิด Timer Interrupt เมื่อหมด Time Slice Process ในสถานะ RUNNING จะถูกเปลี่ยนเป็น RUNNABLE การขัดจังหวะ Process ในขณะรันแบบนี้เรียกว่า Preemption
- 'T' = STOPPED
 - จากสถานะ RUNNABLE process อาจถูก suspend หรือหยุดทำงานชั่วคราว และเปลี่ยนไปอยู่ในสถานะ STOPPED (ทำได้โดยส่งสัญญาณ SIGSTOP หรือกด Ctrl+z)
 - จากสถานะ STOPPED process สามารถเปลี่ยนกลับมาเป็น RUNNABLE (ทำได้ส่งสัญญาณ SIGCONT หรือใช้คำสั่ง “fg”)

<https://www.cbt nuggets.com/blog/certifications/open-source/what-are-the-5-linux-process-states>





Linux Process State



ปรับปรุงจาก

[https://www.thegeeksearch.com/
beginners-guide-to-managing-processes-running-on-a-linux-system/](https://www.thegeeksearch.com/beginners-guide-to-managing-processes-running-on-a-linux-system/)





สถานะ

■ 'S' = INTERRUPTABLE_SLEEP

- เมื่อ Process รอข้อมูลจากเหตุการณ์ใดเหตุการณ์หนึ่ง เช่น (1) รอข้อมูลจากอุปกรณ์ input จาก keyboard และ (2) รอข้อมูลจาก Network Interface Card (NIC) ในสถานะการณ์เหล่านี้ OS kernel จะเปลี่ยนสถานะของ Process ให้เป็น INTERRUPTABLE_SLEEP หรือ 'S'
- ในสถานะ S นี้ Process สามารถถูกขัดจังหวะด้วยสัญญาณ Signal ได้ (รวมทั้งสามารถถูกส่งให้ทำการประมวลผลโดย Signal ได้ด้วย)
- หมายเหตุ: สัญญาณ Signal เป็นการสื่อสารเพื่อขัดจังหวะการประมวลผลของ Process ได Process หนึ่ง แต่ สัญญาณ Interrupt เป็นสัญญาณที่ขัดจังหวะการประมวลผลของ CPU

<https://www.cbt nuggets.com/blog/certifications/open-source/what-are-the-5-linux-process-states>





สถานะ

■ 'S' = INTERRUPTABLE_SLEEP

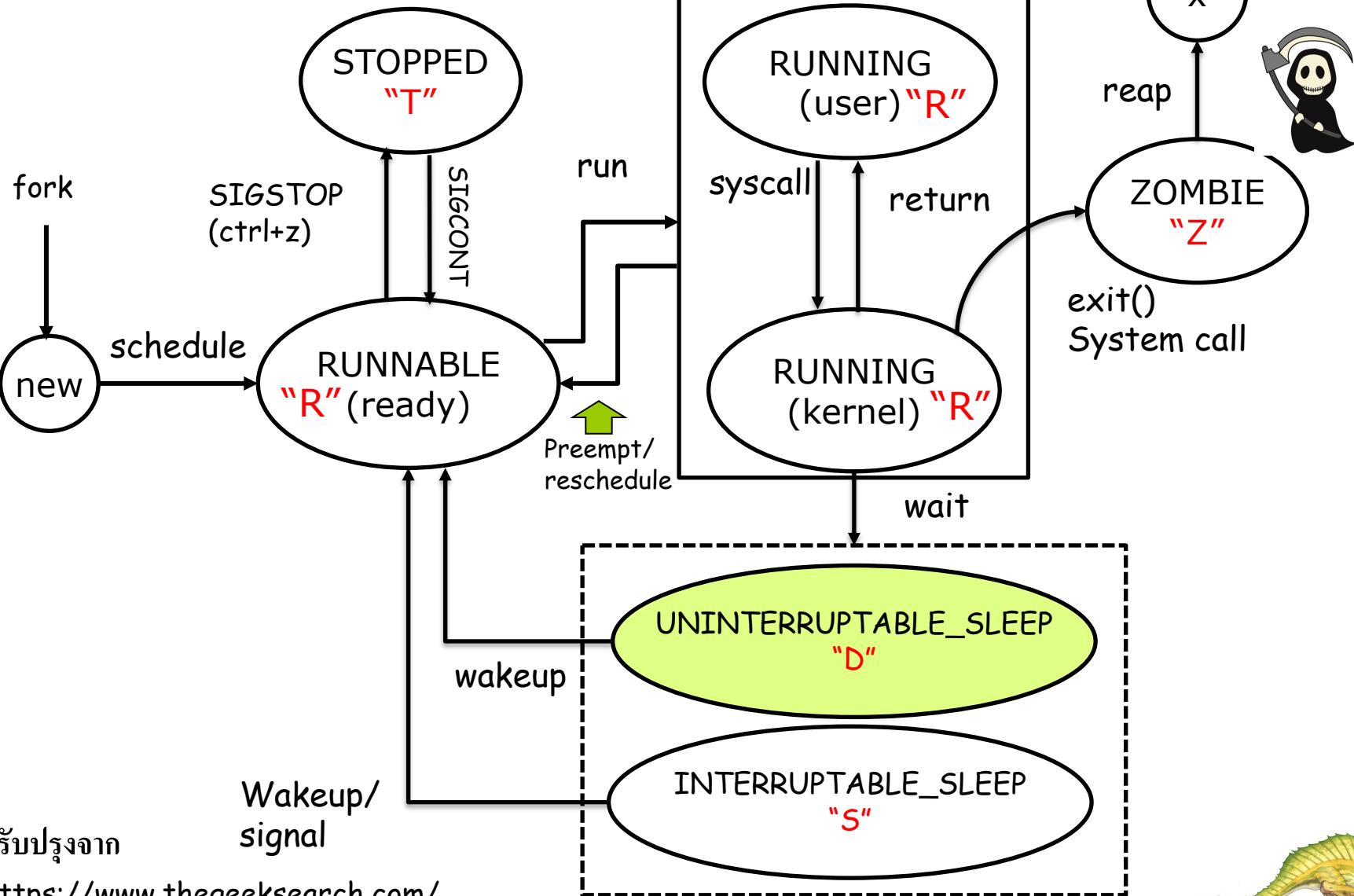
- Process ที่ถูกขัดจังหวะสามารถเลือกได้ว่าจะ รัน function ใน process ที่ถูกเขียนขึ้นเพื่อตอบสนองกับสัญญาณนั้น หรือใช้ default function ของ OS เมื่อรัน function เสร็จแล้ว Process จะ return จาก system call ที่ทำให้เกิดการรอ และประมวลผลคำสั่งถัดไปจากคำสั่งที่ถูกขัดจังหวะก่อนหน้า
- มักเกิดขึ้นเมื่อ (1) รอ I/O device เช่น keyboard (2) รอข้อมูลจาก Network และรอ Slow system calls
- **สำคัญ:** Linux จะ ไม่ใช้ 'S' กับการรอการทำ I/O ที่เกี่ยวกับการอ่านและเขียนข้อมูลจาก SSD หรือ Disk เนื่องจากการขัดจังหวะอาจทำให้การเก็บข้อมูลบน Storage เสียหายและส่งผลให้อุปกรณ์เก็บข้อมูลไม่เสียร

<https://www.cbt nuggets.com/blog/certifications/open-source/what-are-the-5-linux-process-states>





Linux Process State



ปรับปรุงจาก

<https://www.thegeeksearch.com/beginners-guide-to-managing-processes-running-on-a-linux-system/>

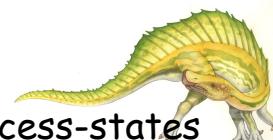




สถานะ

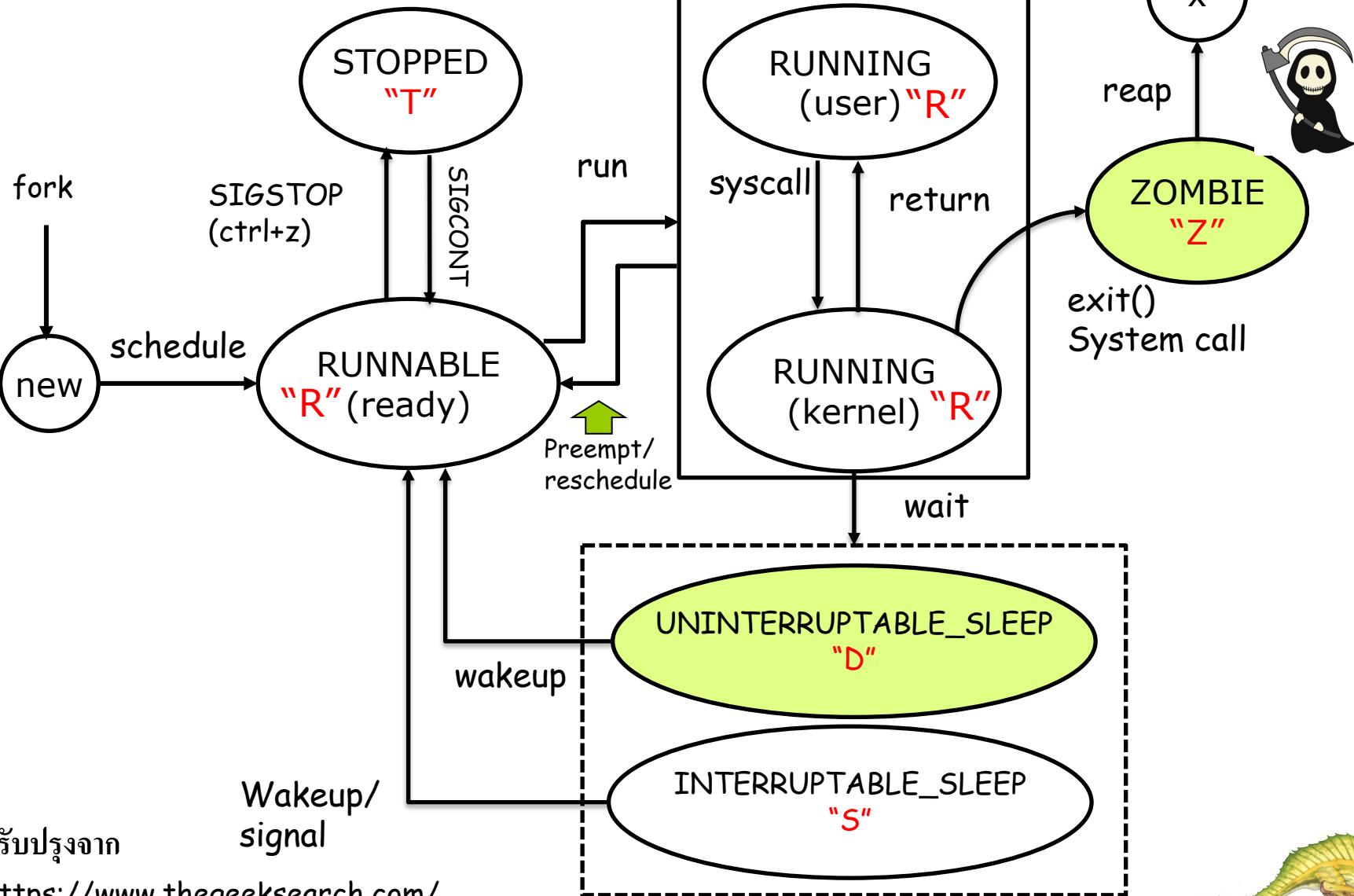
■ 'D' = UNINTERRRUPTABLE_SLEEP

- เมื่อ Process จำเป็นต้องรอข้อมูลจากเหตุการณ์ที่ไม่สามารถถูกขัดจังหวะได้ เพราะการขัดจังหวะจะทำให้เกิด OS error ดังนั้นจึงต้องรอนกว่าจะทำงาน system call ที่ทำให้เกิดการรอเสร็จ หรือจน OS ตัดสินว่าทำไม่สำเร็จและรีเทิร์นพร้อมกับแจ้งข้อมูล Error ขึ้น
- การรอแบบนี้ได้แก่ (1) การอ่านและเขียนข้อมูลบนระบบ Storage SSD และ Disks เช่น คำสั่ง “mkdir” ในระหว่างที่คำสั่งนี้ทำงาน Process จะอยู่ใน สถานะ ‘D’ ไม่สามารถขัดจังหวะได้ มีแค่สำเร็จหรือ error เท่านั้น
- (2) การอ่านข้อมูลจาก Network File System (NFS) ที่ระหว่างอ่านข้อมูลจาก remote server ถ้ามีปัญหา network ระบบจะต้องรอนกว่า network จะกลับมาเป็นปกติ หรือถ้ารอนานมากๆ จะเกิด Timeout และระบบจะแจ้ง error





Linux Process State



ปรับปรุงจาก

[https://www.thegeeksearch.com/
beginners-guide-to-managing-processes-running-on-a-linux-system/](https://www.thegeeksearch.com/beginners-guide-to-managing-processes-running-on-a-linux-system/)





สถานะ

- 'D' = UNINTERRRUPTABLE_SLEEP

- ในสถานะนี้ Process จะไม่สามารถถูกขัดจังหวะด้วยสัญญาณ Signal (Interrupt) ได้

- 'Z' = ZOMBIE

- เป็นสถานะที่ Process จบการทำงานด้วย exit() system call แล้ว แต่ ข้อมูลสถานะของ Process ยังคงอยู่ใน Process Table และยังไม่มีการ clear ค่าของ Process นั้นใน Process Table
 - เป็นหน้าที่ของ Parent Process ที่ต้องมาอ่านค่าและ clear ค่าสถานะนี้ (เรียกอีกอย่างว่าการ reap = ตัดด้วยเคียว) ด้วยคำสั่งเช่น wait, waitpid

<https://www.cbt nuggets.com/blog/certifications/open-source/what-are-the-5-linux-process-states>





สถานะ

■ 'Z' = ZOMBIE

- เป็นสถานะที่ Child Process จะการประมวลผล และ **parent process** ยังไม่จบการประมวลผล **แต่ parent ยังไม่**เรียก `wait()` หรือ `waitpid()`
- ถ้า Parent Process จะการประมวลผล Zombie Process จะถูกกำจัดโดย INIT โดยอัตโนมัติ

<https://www.cbt nuggets.com/blog/certifications/open-source/what-are-the-5-linux-process-states>





การพัฒนาโปรแกรมภาษา C

- ในการพัฒนาโปรแกรมขนาดใหญ่ โปรแกรมเมอร์มักจะสร้างไฟล์หลายไฟล์ และคอมไฟล์ไฟล์เหล่านั้นแยกกัน เนื่องจากไฟล์แต่ละไฟล์มีโค้ดที่เกี่ยวข้อง กับเรื่องใดเรื่องหนึ่งโดยเฉพาะ
- นอกจากนี้ ในการนิ่มผู้พัฒนาโปรแกรมหลายคนแต่ละคนอาจพัฒนาไฟล์ แยกกัน
- แต่เมื่อจะสร้าง executable file จะต้องนำไฟล์เหล่านั้นมา link ร่วมกัน
- วิธีการพื้นฐานในการพัฒนาโปรแกรมภาษาซีอย่างหนึ่งคือการใช้ make
- ในการคอมไไฟล์โปรแกรม เช่น qemu-kvm บนลินุกซ์ใช้วิธีนี้ร่วมกับ tool อื่น





ติดตั้ง C compiler และ compile C program

```
$ sudo apt install gcc make  
(or apt install build-essentials)  
$ nano hello.c  
#include <stdio.h>  
main(){  
    printf("hello world :)\n");  
}  
$ gcc hello.c  
$ ./a.out  
$ gcc -o hello hello.c  
$ ./hello
```





การคอมไพล์ object code และ link โปรแกรม (1)

```
$ mkdir myprograms  
$ cd myprograms  
$ nano hellosub1.c  
#include <stdio.h>  
#include "hello1.h"  
void fun1(void){  
    printf("hello1 %d\n", CONSTANT1);  
}  
$ gcc -c hellosub1.c
```





การคอมไพล์ object code และ link โปรแกรม (1)

```
$ nano hello1.h
$ cat hello1.h
#define CONSTANT1 100
$
```





การคอมไพล์ object code และ link โปรแกรม (2)

```
$ nano hellosub2.c
#include <stdio.h>

void fun2(void){
    printf("hello2\n");
}

$ gcc -c hellosub2.c
$ ls -l
```





การคอมไพล์ object code และ link โปรแกรม (3)

```
$ nano hellomain.c
#include <stdio.h>
extern void fun1(void);
extern void fun2(void);

int main(){
    fun1();
    fun2();
}

$ gcc -c hellomain.c
$ ls -l
```





การคอมไพล์ object code และ link โปรแกรม (4)

```
$ gcc -o hello hellomain.o \
> hellosub1.o hellosub2.o
$ ls -l
$ file *
$ ./hello
```



Process Control

- Fork, Exec, and Waitpid
- Fork spawn a new process
- Exec load executable to a process
- waitpid waits until a child process finishes execution

pid-related system calls

```
#include <unistd.h>

pid_t getpid(void);
```

Returns: process ID of calling process

```
pid_t getppid(void);
```

Returns: parent process ID of calling process

```
uid_t getuid(void);
```

Returns: real user ID of calling process



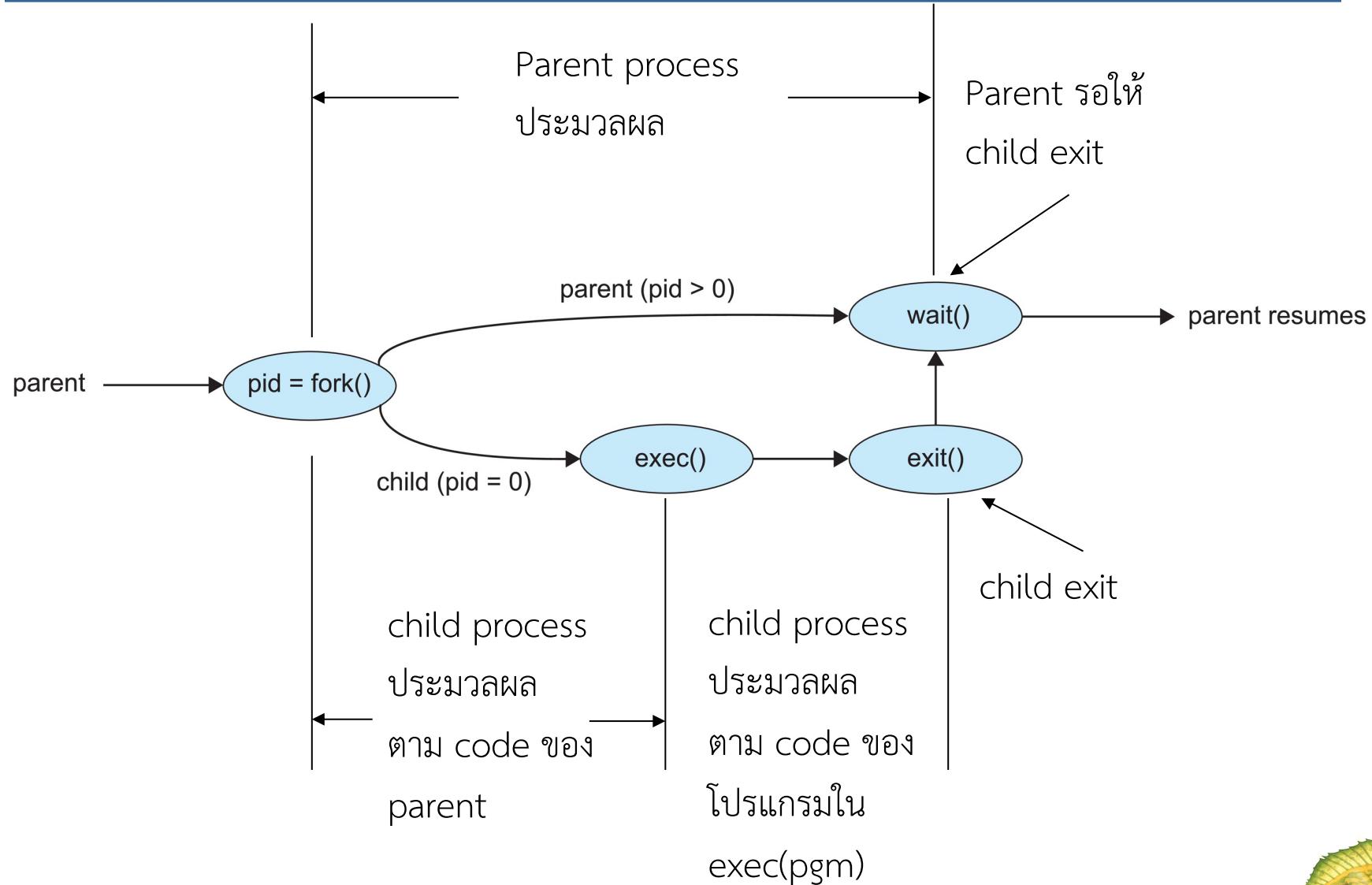
Process creation in Linux/UNIX

- Fork create a new process, a copy of the caller
- The caller is a parent process
- The new process is a child process
- fork called once but return twice
- It return PID on the parent and return 0 on the child





Process Creation (Cont.)



fork() system call

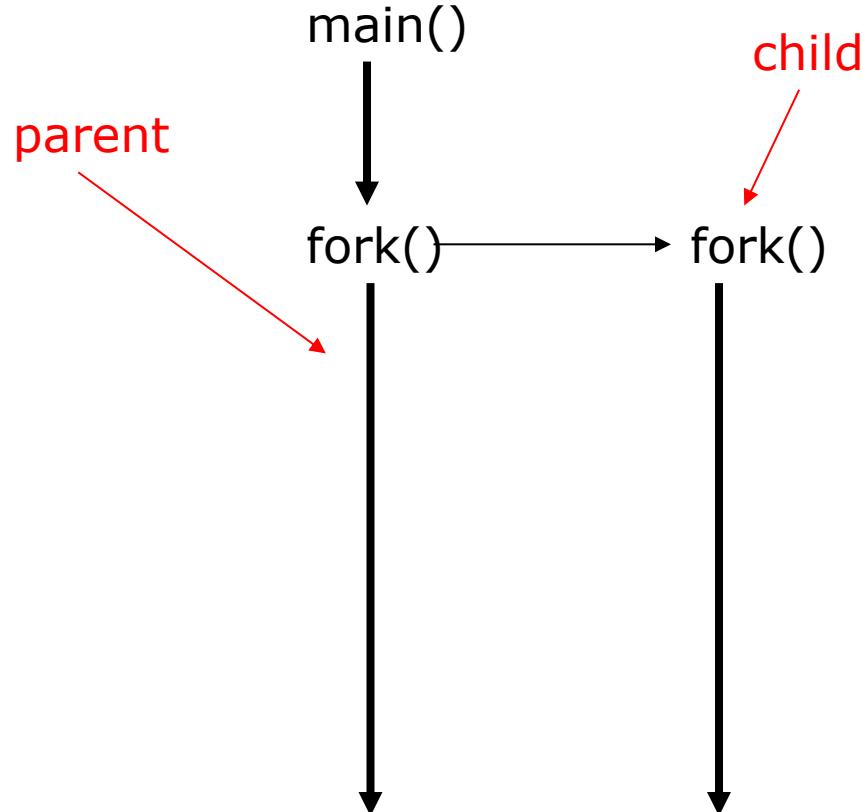
```
#include <unistd.h>

pid_t fork(void);
```

Returns: 0 in child, process ID of child in parent, -1 on error

- สร้าง process ใหม่
- Child ทำสำเนา data heap and stack segment ของ parent
- แยก memory space

fork



Fork เป็น system call ที่ทำหน้าที่สร้าง process ใหม่ (child process) ด้วยการ Clone process คือ copy คุณสมบัติของ parent และนำมาสร้างเป็นของตนเอง ได้แก่

- สถานะการทำงาน CPU state
- เนื้อหา Memory contents

และใช้ของร่วมกันกับ parent ได้แก่

- Open files ต่างๆ

รันจนจบการทำงาน หรือได้รับ termination signal



ex01.c

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <inttypes.h>

int main(int argc, char * argv[])
{
    pid_t    pid;
    int      status;

    pid = getpid();
    printf("before call fork(), pid = %ld\n", (long)pid);
    fflush(stdout);

    fork();

    pid = getpid();
    printf("after call fork(), pid = %ld\n", (long)pid);
    fflush(stdout);

    exit(0);
}
```



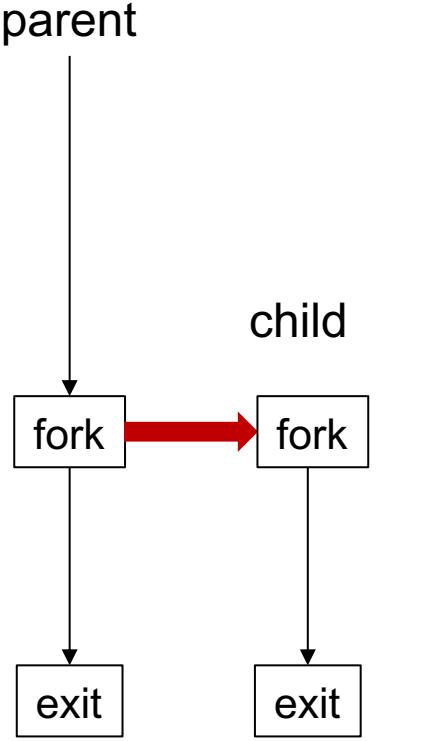


ໃໝ່ ນສ Visualize ໂປຣແກຣມ ex01.c

```
int main(int argc, char * argv[]) parent
{
    pid_t    pid;
    int      status;

    pid = getpid();
    printf("before call fork(),
fflush(stdout);

    fork();
```



```
    fork();

    pid = getpid();
    printf("after call fork(),
fflush(stdout);

    exit(0);
```





“after call อันไหนเป็นของ child”

```
[u2909610384@cs22201:~/ch3$ gcc -o ex01 expgm01.c
[u2909610384@cs22201:~/ch3$ ./ex01
before call fork(), pid = 124749
after call fork(), pid = 124749
after call fork(), pid = 124750
```





ex02.c

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <inttypes.h>

int main(int argc, char * argv[])
{
    pid_t    pid;
    int      status;

    pid = getpid();
    printf("before call fork(), pid = %ld\n", (long)pid);
    fflush(stdout);

    fork();
    fork();

    pid = getpid();
    printf("after call fork(), pid = %ld\n", (long)pid);
    fflush(stdout);

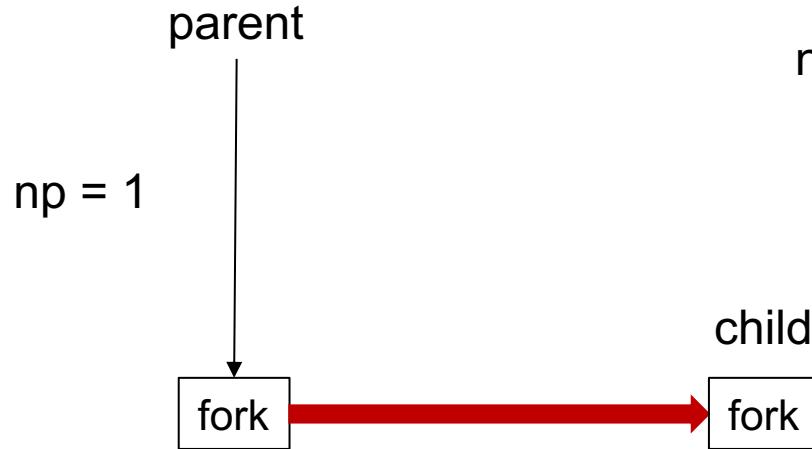
    exit(0);
}
```





ໃໝ່ ນສ Visualize ໂປຣແກຣມ ex02.c

```
int main(int argc, char *  
{  
    pid_t    pid;  
    int      status;  
    pid = getpid();  
    printf("before ca  
fflush(stdout);  
  
    fork();  
}  
  
np = 1  
  
fork();  
  
pid = getpid();  
printf("after ca]  
fflush(stdout);  
  
exit(0);
```



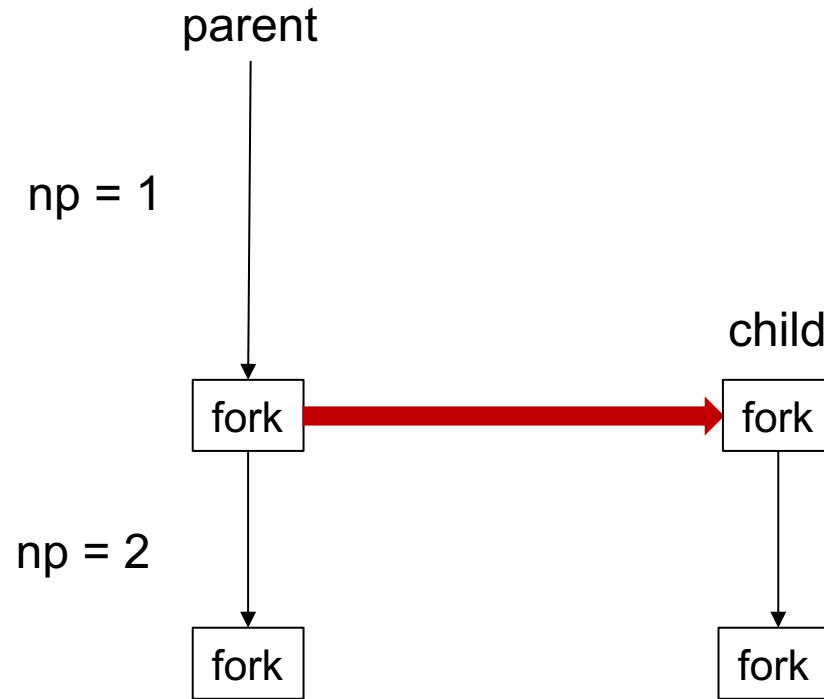
np = ຈຳນວນ process





ໃໝ່ ນສ Visualize ໂປຣແກຣມ ex02.c

```
int main(int argc, char *  
{  
    pid_t    pid;  
    int      status;  
    pid = getpid();  
    printf("before ca  
fflush(stdout);  
  
    fork();  
  
    np = 2  
  
    fork();  
  
    pid = getpid();  
    printf("after ca]  
fflush(stdout);  
  
    exit(0);  
}
```





ให้ นศ Visualize โปรแกรม ex02.c

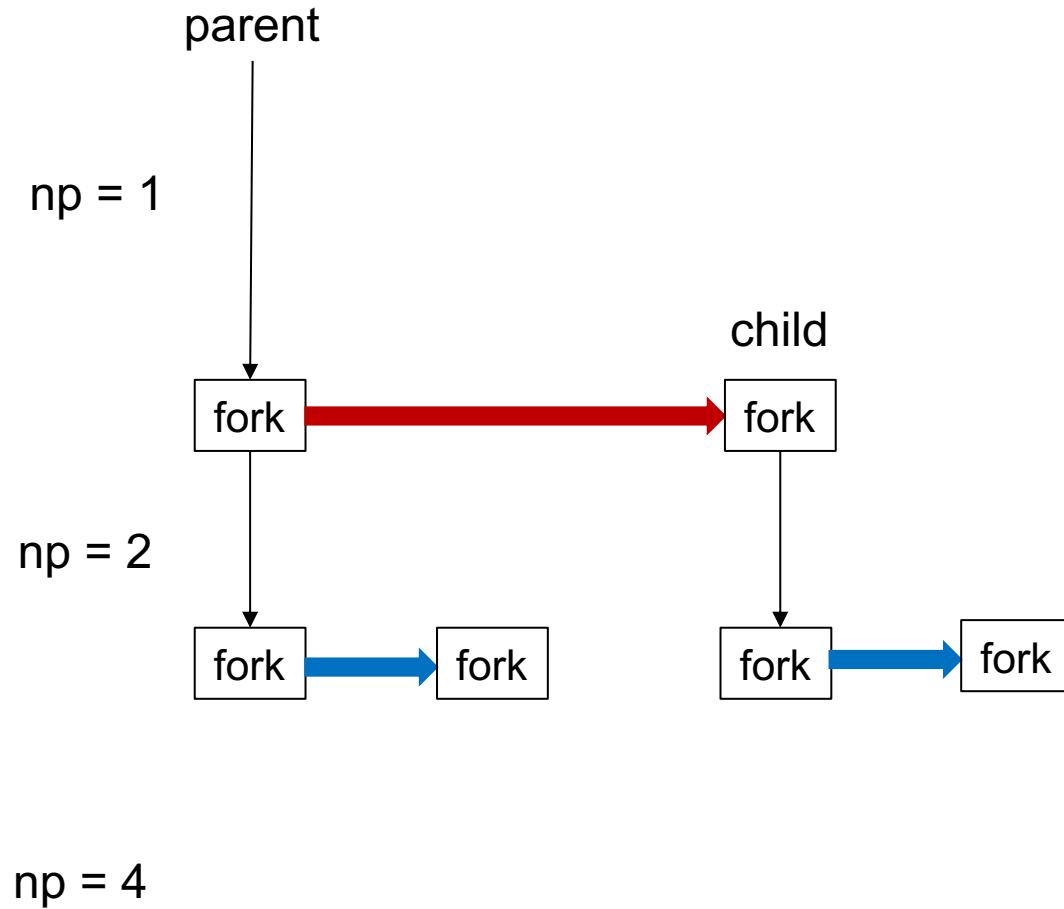
```
int main(int argc, char *argv)
{
    pid_t pid;
    int status;

    pid = getpid();
    printf("before call\n");
    fflush(stdout);

    fork();

    pid = getpid();
    printf("after call\n");
    fflush(stdout);

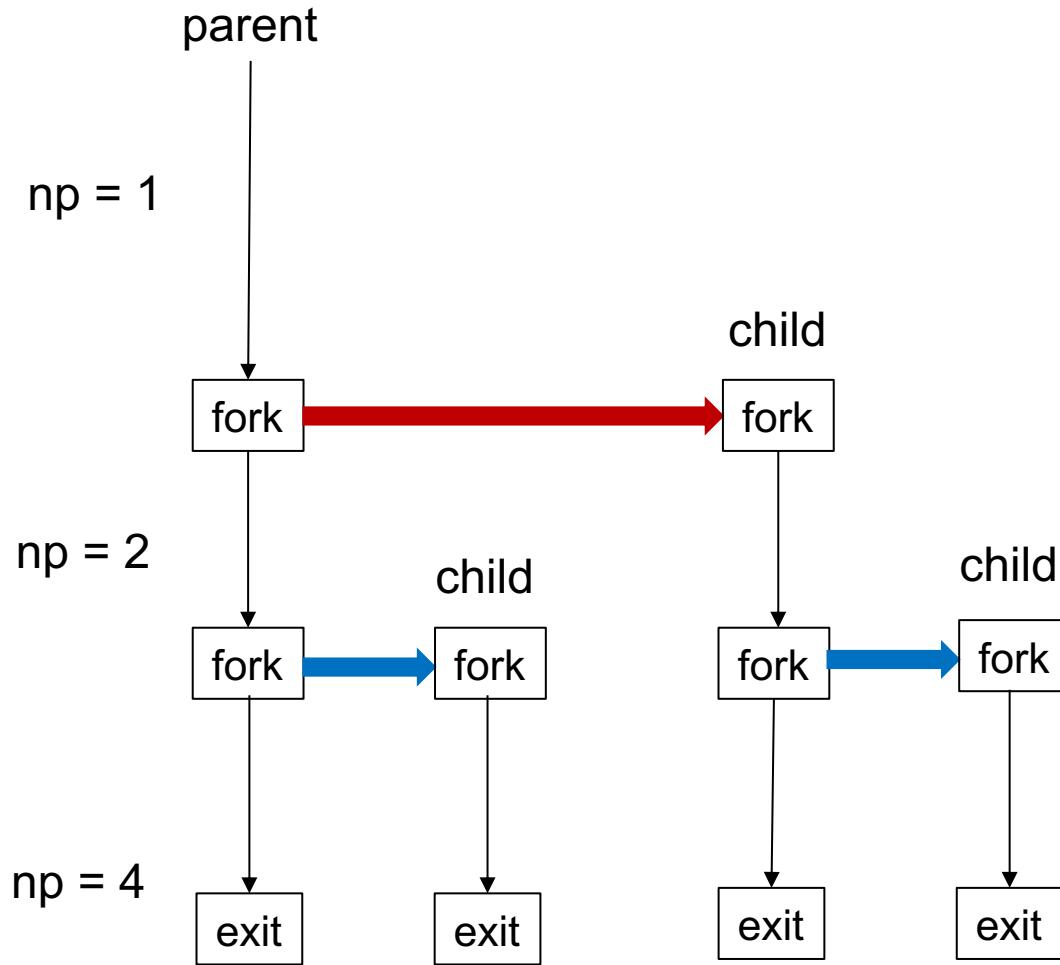
    exit(0);
}
```





ໃໝ່ ນສ Visualize ໂປຣແກຣມ ex02.c

```
int main(int argc, char *  
{  
    pid_t    pid;  
    int      status;  
    pid = getpid();  
    printf("before ca  
fflush(stdout);  
  
    fork();  
  
    fork();  
  
    pid = getpid();  
    printf("after ca]  
fflush(stdout);  
  
    exit(0);  
}
```





Binary Tree

Full Binary Tree



A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children.

It is also known as **a proper binary tree**.

<https://www.programiz.com/dsa/full-binary-tree>

1. The maximum number of nodes at level 'l' of a binary tree is 2^l :

Note: Here level is the number of nodes on the path from the root to the node (including root and node). The level of the root is 0

This can be proved by induction:

For root, $l = 0$, number of nodes = $2^0 = 1$

Assume that the maximum number of nodes on level 'l' is 2^l

*Since in a Binary tree every node has at most 2 children, the next level would have twice nodes, i.e. $2 * 2^l$*

<https://www.geeksforgeeks.org/properties-of-binary-tree/>





```
[u2909610384@cs22201:~/ch3$ gcc -o ex02 expgm02.c
[u2909610384@cs22201:~/ch3$ 
[u2909610384@cs22201:~/ch3$ ./ex02
before call fork(), pid = 124761
after call fork(), pid = 124761
after call fork(), pid = 124762
after call fork(), pid = 124763
after call fork(), pid = 124764
[u2909610384@cs22201:~/ch3$
```





ex2-gen.c (ให้ printf PID และ PPID)

```
int main(int argc, char * argv[])
{
    int i; int n=8;

    for (i=0; i< n; i++) fork();

    exit(0);
}
```

การบ้าน ให้ นศ Visualize โปรแกรม ex02-gen.c เมื่อ n = 4
ให้ระบุสูตรคำนวณ np ในแต่ละชั้นด้วย กำหนดให้เขียน np เป็นพังก์ชันของ n
หรือ np = f(n)



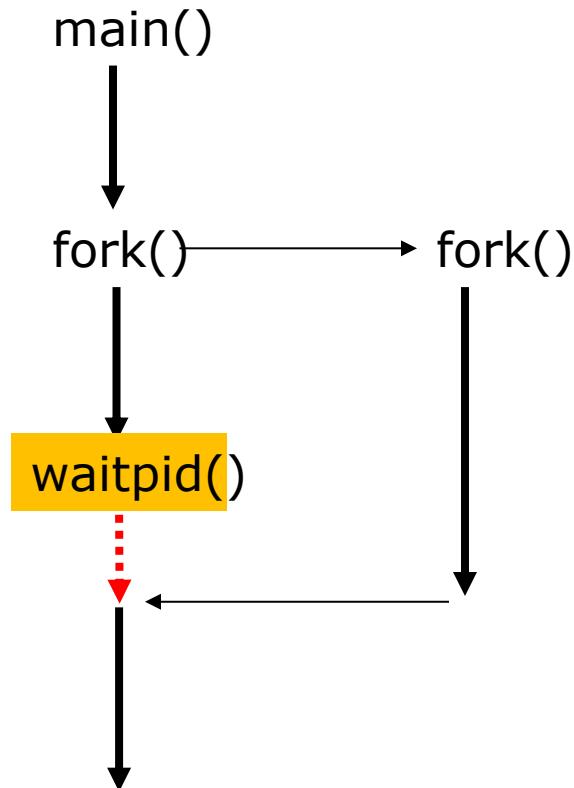


```
1 #include <stdio.h>
2 #include <signal.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <string.h>
6 #include <sys/wait.h>
7 #include <inttypes.h>
8
9 int main(int argc, char * argv[])
10 {
11     pid_t    pid;
12     int      status;
13     int i; int n=8;
14
15     pid = getpid();
16     printf("before call fork(), pid = %ld\n",
17            (long)pid);
18     fflush(stdout);
19
20     for (i=0; i< n; i++) fork();
21
22     pid = getpid();
23     printf("after call fork(), pid = %ld\n",
24            (long)pid);
25     fflush(stdout);
26     exit(0);
27 }
```

```
u2909610384@cs22201:~/ch3/Solutions$ gcc -o fk fork.c
u2909610384@cs22201:~/ch3/Solutions$ ./fk > output.txt
u2909610384@cs22201:~/ch3/Solutions$ head -5 output.txt
before call fork(), pid = 221389
after call fork(), pid = 221391
after call fork(), pid = 221393
after call fork(), pid = 221396
after call fork(), pid = 221395
u2909610384@cs22201:~/ch3/Solutions$ tail -4 output.txt
after call fork(), pid = 221642
after call fork(), pid = 221643
after call fork(), pid = 221641
after call fork(), pid = 221644
u2909610384@cs22201:~/ch3/Solutions$ grep after output.txt | wc -l
256
u2909610384@cs22201:~/ch3/Solutions$
```



fork and waitpid



รันจนจบการทำงาน



Wait and waitpid

```
#include <sys/wait.h>

pid_t wait(int *statloc);

pid_t waitpid(pid_t pid, int *statloc, int options);
```

Both return: process ID if OK, 0 (see later), or -1 on error





ex03.c

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <inttypes.h>

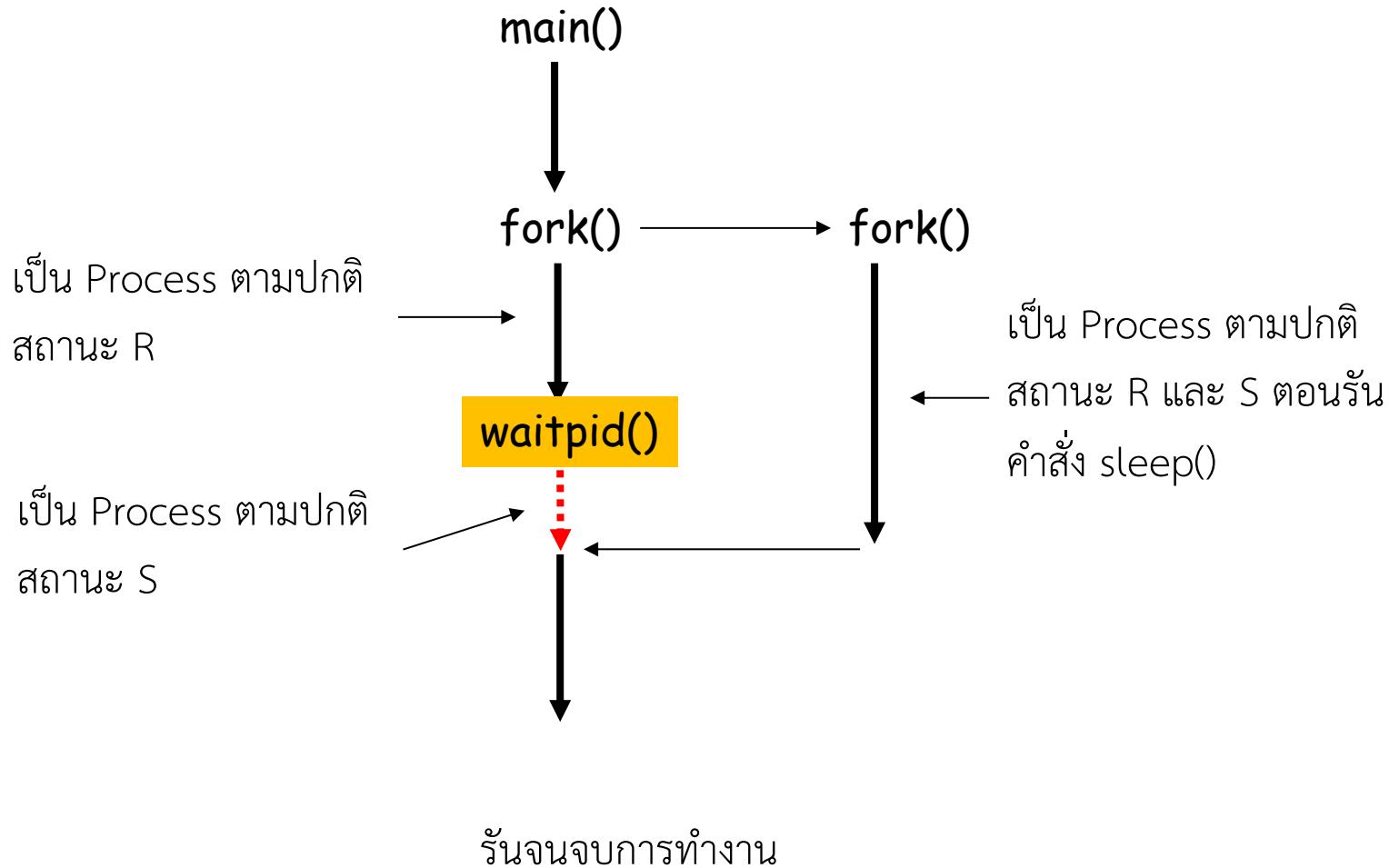
int main(int argc, char * argv[])
{
    pid_t    pid;
    int      status;

    if ((pid = fork()) < 0) {
        printf("fork error");
        exit(1);
    }
    else if (pid == 0) { /* child */
        printf("This is a child process\n");
        sleep(4);
        printf("Child process terminate\n");
        exit(0);
    }
    else{
        /* parent */
        waitpid(pid, &status, 0);
    }
    exit(0);
}
```





See Process State



รันจนจบการทำงาน





Orphan Process (ex4.c)

```
 9 int main(int argc, char * argv[])
10 {
11     pid_t    pid;
12     pid_t    mypid;
13     int      status;
14
15     if ((pid = fork()) < 0) {
16         exit(1);
17     }
18     else if (pid == 0) {
19         mypid = getpid();
20         printf("child pid = %ld\n", (long)mypid); fflush(stdout);
21         sleep(250);
22         exit(0);
23     }
24     else{
25         mypid = getpid();
26         printf("parent pid = %ld\n", (long)mypid); fflush(stdout);
27         sleep(2);
28         waitpid(pid, &status, 0);
29         exit(0);
30     }
31     exit(0);
32 }
```





```
u2909610384@cs22201:~/ch3/Solutions$ ps -fu
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
u290961+  221657  0.0  0.0  10148  5328 pts/2      Ss  06:35  0:00 -bash
u290961+  221730  0.0  0.0   2488   580 pts/2      S+  06:52  0:00 \_ ./nz
u290961+  221731  0.0  0.0   2488    76 pts/2      S+  06:52  0:00 \_ ./nz
u290961+  220379  0.0  0.0  10016  5264 pts/1      Ss  06:06  0:00 -bash
u290961+  221732  0.0  0.0  10612  3408 pts/1      R+  06:52  0:00 \_ ps -fu
u290961+  220238  0.0  0.0  10128  5576 pts/0      Ss  05:52  0:00 -bash
u290961+  220376  0.0  0.0   7136  3576 pts/0      S+  06:06  0:00 \_ tmux
u2909610384@cs22201:~/ch3/Solutions$
```

```
u2909610384@cs22201:~/ch3/Solutions$ gcc -o nz nozombie.c
u2909610384@cs22201:~/ch3/Solutions$ ./nz
parent pid = 221730
child pid = 221731
```





```
u2909610384@cs22201:~/ch3/Solutions$ ps -fu
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
u290961+  221657  0.0  0.0  10148  5328 pts/2      Ss  06:35  0:00 -bash
u290961+  221730  0.0  0.0   2488   580 pts/2      S+  06:52  0:00 \_ ./nz
u290961+  221731  0.0  0.0   2488    76 pts/2      S+  06:52  0:00 \_ ./nz
u290961+  220379  0.0  0.0  10016  5264 pts/1      Ss  06:06  0:00 -bash
u290961+  221732  0.0  0.0  10612  3408 pts/1      R+  06:52  0:00 \_ ps -fu
u290961+  220238  0.0  0.0  10128  5576 pts/0      Ss  05:52  0:00 -bash
u290961+  220376  0.0  0.0    7136  3576 pts/0      S+  06:06  0:00 \_ tmux
u2909610384@cs22201:~/ch3/Solutions$ kill -9 221730
u2909610384@cs22201:~/ch3/Solutions$
u2909610384@cs22201:~/ch3/Solutions$
```

```
u2909610384@cs22201:~/ch3/Solutions$ gcc -o nz nozombie.c
u2909610384@cs22201:~/ch3/Solutions$ ./nz
parent pid = 221730
child pid = 221731
Killed
u2909610384@cs22201:~/ch3/Solutions$
```





```
u2909610384@cs22201:~/ch3/Solutions$ ps -fu
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
u290961+  221657  0.0  0.0  10148  5328 pts/2      Ss+  06:35  0:00 -bash
u290961+  220379  0.0  0.0  10016  5264 pts/1      Ss  06:06  0:00 -bash
u290961+  221734  0.0  0.0  10612  3336 pts/1      R+  06:54  0:00 \_ ps -fu
u290961+  220238  0.0  0.0  10128  5576 pts/0      Ss  05:52  0:00 -bash
u290961+  220376  0.0  0.0   7136  3576 pts/0      S+  06:06  0:00 \_ tmux
u290961+  221731  0.0  0.0   2488     76 pts/2      S  06:52  0:00 ./nz
u2909610384@cs22201:~/ch3/Solutions$ ps -ef | head -1
UID          PID  PPID  C STIME TTY          TIME CMD
u2909610384@cs22201:~/ch3/Solutions$ ps -ef | grep nz
u290961+  221731      1  0 06:52 pts/2  00:00:00 ./nz
u290961+  221738  220379  0 06:54 pts/1  00:00:00 grep --color=auto nz
u2909610384@cs22201:~/ch3/Solutions$
u2909610384@cs22201:~/ch3/Solutions$ █
```

```
u2909610384@cs22201:~/ch3/Solutions$ gcc -o nz nozombie.c
u2909610384@cs22201:~/ch3/Solutions$ ./nz
parent pid = 221730
child pid = 221731
Killed
u2909610384@cs22201:~/ch3/Solutions$
```





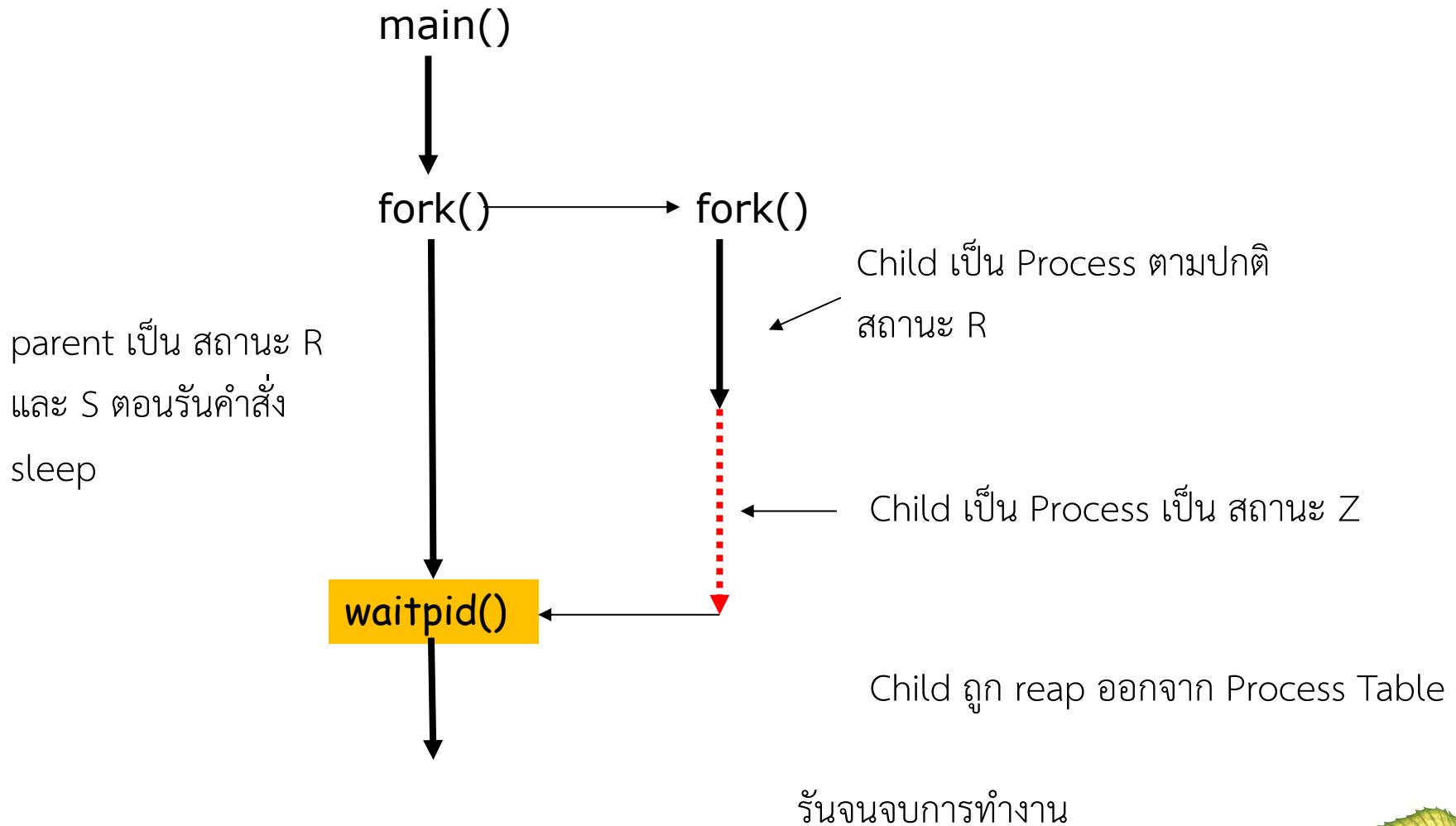
Zombie Process (ex5.c)

```
 9 int main(int argc, char * argv[])
10 {
11     pid_t    pid;
12     int      status;
13
14     if ((pid = fork()) < 0) {
15         exit(1);
16     }
17     else if (pid == 0) {
18         sleep(4);
19         exit(0);
20     }
21     else{
22         sleep(25);
23         waitpid(pid, &status, 0);
24         exit(0);
25     }
26     exit(0);
27 }
```





See Process State





ex5-1.c

```
 9 int main(int argc, char * argv[])
10 {
11     pid_t    pid;
12     pid_t    mypid;
13     int      status;
14
15     if ((pid = fork()) < 0) {
16         exit(1);
17     }
18     else if (pid == 0) {
19         mypid = getpid();
20         printf("child pid = %ld\n", (long)mypid); fflush(stdout);
21         sleep(4);
22         exit(0);
23     }
24     else{
25         mypid = getpid();
26         printf("parent pid = %ld\n", (long)mypid); fflush(stdout);
27         for(;;)
28             //sleep(250);
29             waitpid(pid, &status, 0);
30             exit(0);
31     }
32     exit(0);
33 }
```





```
u2909610384@cs22201:~/ch3/Solutions$  
u2909610384@cs22201:~/ch3/Solutions$ ps -fu  
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND  
u290961+  221747  0.0  0.0  10148  5300 pts/1      Ss  07:08  0:00 -bash  
u290961+  221768 97.7  0.0   2488   516 pts/1      R+  07:09  0:18 \_ ./z1  
u290961+  221769  0.0  0.0     0     0 pts/1      Z+  07:09  0:00 \_ [z1] <defunct>  
u290961+  221657  0.0  0.0  10148  5328 pts/2      Ss  06:35  0:00 -bash  
u290961+  221772  0.0  0.0  10612  3452 pts/2      R+  07:09  0:00 \_ ps -fu  
u290961+  220238  0.0  0.0  10128  5576 pts/0      Ss  05:52  0:00 -bash  
u290961+  220376  0.0  0.0   7136  3576 pts/0      S+  06:06  0:00 \_ tmux  
u2909610384@cs22201:~/ch3/Solutions$  
u2909610384@cs22201:~/ch3/Solutions$ █
```

```
u2909610384@cs22201:~/ch3/Solutions$ gcc -o z1 zombie1.c  
u2909610384@cs22201:~/ch3/Solutions$ ./z1  
parent pid = 221768  
child pid = 221769
```





```
u290961+ 221769 0.0 0.0      0      0 pts/1    Z+  07:09  0:00      \_ [z1] <defunct>
u290961+ 221657 0.0 0.0 10148  5328 pts/2    Ss  06:35  0:00 -bash
u290961+ 221772 0.0 0.0 10612  3452 pts/2    R+  07:09  0:00 \_ ps -fu
u290961+ 220238 0.0 0.0 10128  5576 pts/0    Ss  05:52  0:00 -bash
u290961+ 220376 0.0 0.0    7136  3576 pts/0    S+  06:06  0:00 \_ tmux
u2909610384@cs22201:~/ch3/Solutions$ kill -9 221768
u2909610384@cs22201:~/ch3/Solutions$ ps -fu
USER          PID %CPU %MEM    VSZ   RSS TTY STAT START   TIME COMMAND
u290961+ 221747 0.0 0.0 10148  5300 pts/1    Ss+ 07:08  0:00 -bash
u290961+ 221657 0.0 0.0 10148  5328 pts/2    Ss  06:35  0:00 -bash
u290961+ 221774 0.0 0.0 10612  3364 pts/2    R+  07:13  0:00 \_ ps -fu
u290961+ 220238 0.0 0.0 10128  5576 pts/0    Ss  05:52  0:00 -bash
u290961+ 220376 0.0 0.0    7136  3576 pts/0    S+  06:06  0:00 \_ tmux
u2909610384@cs22201:~/ch3/Solutions$
```

```
u2909610384@cs22201:~/ch3/Solutions$ gcc -o z1 zombie1.c
u2909610384@cs22201:~/ch3/Solutions$ ./z1
parent pid = 221768
child pid = 221769
Killed
u2909610384@cs22201:~/ch3/Solutions$
```





ex5-2.c

```
9 int main(int argc, char * argv[])
10 {
11     pid_t    pid;
12     pid_t    mypid;
13     int      status;
14
15     if ((pid = fork()) < 0) {
16         exit(1);
17     }
18     else if (pid == 0) {
19         mypid = getpid();
20         printf("child pid = %ld\n", (long)mypid); fflush(stdout);
21         sleep(4);
22         exit(0);
23     }
24     else{
25         mypid = getpid();
26         printf("parent pid = %ld\n", (long)mypid); fflush(stdout);
27         //for(;;);
28         sleep(250);
29         waitpid(pid, &status, 0);
30         exit(0);
31     }
32     exit(0);
33 }
```





```
u2909610384@cs22201:~/ch3/Solutions$  
u2909610384@cs22201:~/ch3/Solutions$ ps -fu  
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND  
u290961+  221747  0.0  0.0  10148  5300 pts/1      Ss  07:08  0:00 -bash  
u290961+  221782  0.0  0.0   2488   516 pts/1      S+  07:15  0:00 \_ ./z2  
u290961+  221783  0.0  0.0     0     0 pts/1      Z+  07:15  0:00 \_ [z2] <defunct>  
u290961+  221657  0.0  0.0  10148  5328 pts/2      Ss  06:35  0:00 -bash  
u290961+  221784  0.0  0.0  10612  3448 pts/2      R+  07:15  0:00 \_ ps -fu  
u290961+  220238  0.0  0.0  10128  5576 pts/0      Ss  05:52  0:00 -bash  
u290961+  220376  0.0  0.0    7136  3576 pts/0      S+  06:06  0:00 \_ tmux  
u2909610384@cs22201:~/ch3/Solutions$  
u2909610384@cs22201:~/ch3/Solutions$ █
```

```
u2909610384@cs22201:~/ch3/Solutions$ gcc -o z2 zombie2.c  
u2909610384@cs22201:~/ch3/Solutions$  
u2909610384@cs22201:~/ch3/Solutions$ ./z2  
parent pid = 221782  
child pid = 221783
```





ex6.c

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <inttypes.h>

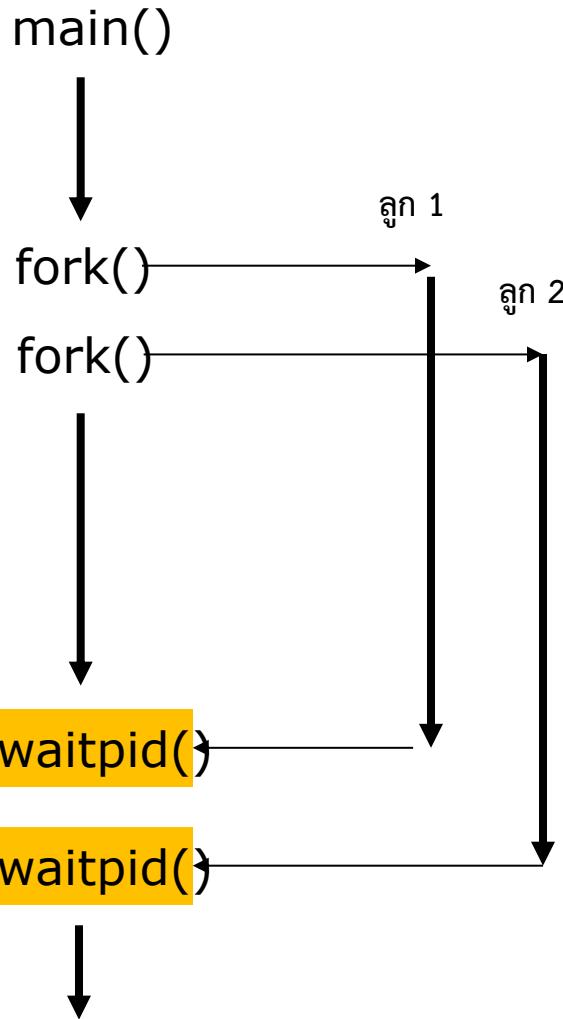
int main(int argc, char * argv[])
{
    pid_t    pid;
    int      status;

    if ((pid = fork()) == 0) {      /* child */
        printf("This is a child process\n");
        sleep(4);
        printf("Child process terminate\n");
        exit(0);
    }

    /* parent */
    waitpid(pid, &status, 0);

    exit(0);
}
```







ex7.c

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <inttypes.h>

int main(int argc, char * argv[])
{
    pid_t    pid1, pid2;
    int      status;
    if ((pid1 = fork()) == 0) { /* child1 */
        printf("This is a child process\n");
        sleep(4);
        printf("Child process terminate\n");
        exit(0);
    }
    if ((pid2 = fork()) == 0) { /* child2 */
        printf("This is a child process\n");
        sleep(4);
        printf("Child process terminate\n");
        exit(0);
    }
    /* parent */
    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);

    exit(0);
}
```



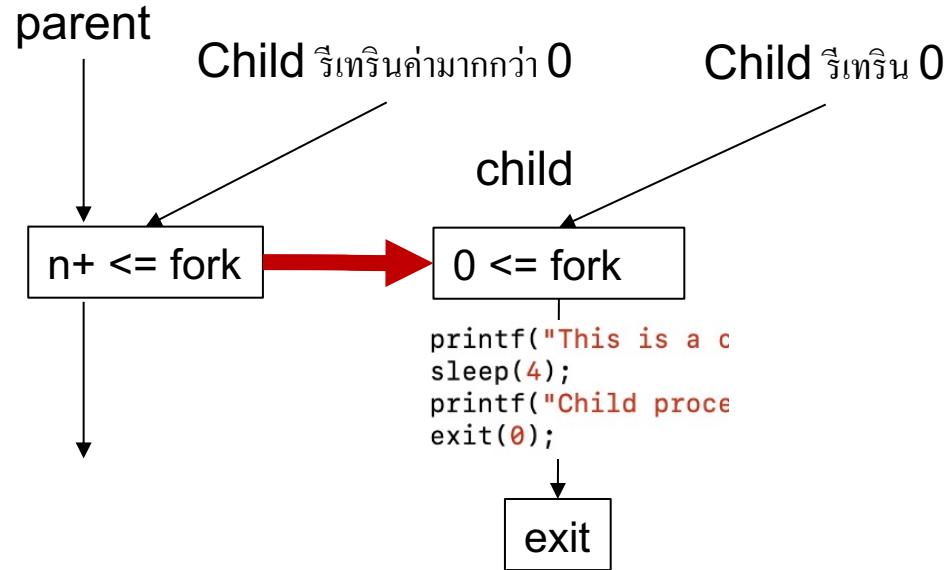


Visualize ex7.c

```
int main(int argc, char * argv[])
{
    pid_t pid1, pid2;
    int status;

    if ((pid1 = fork()) == 0) {
        printf("This is a c
sleep(4);
printf("Child proce
exit(0);
    }
    if ((pid2 = fork()) == 0) {
        printf("This is a c
sleep(4);
printf("Child proce
exit(0);
    }
    /* parent */
    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);

    exit(0);
}
```



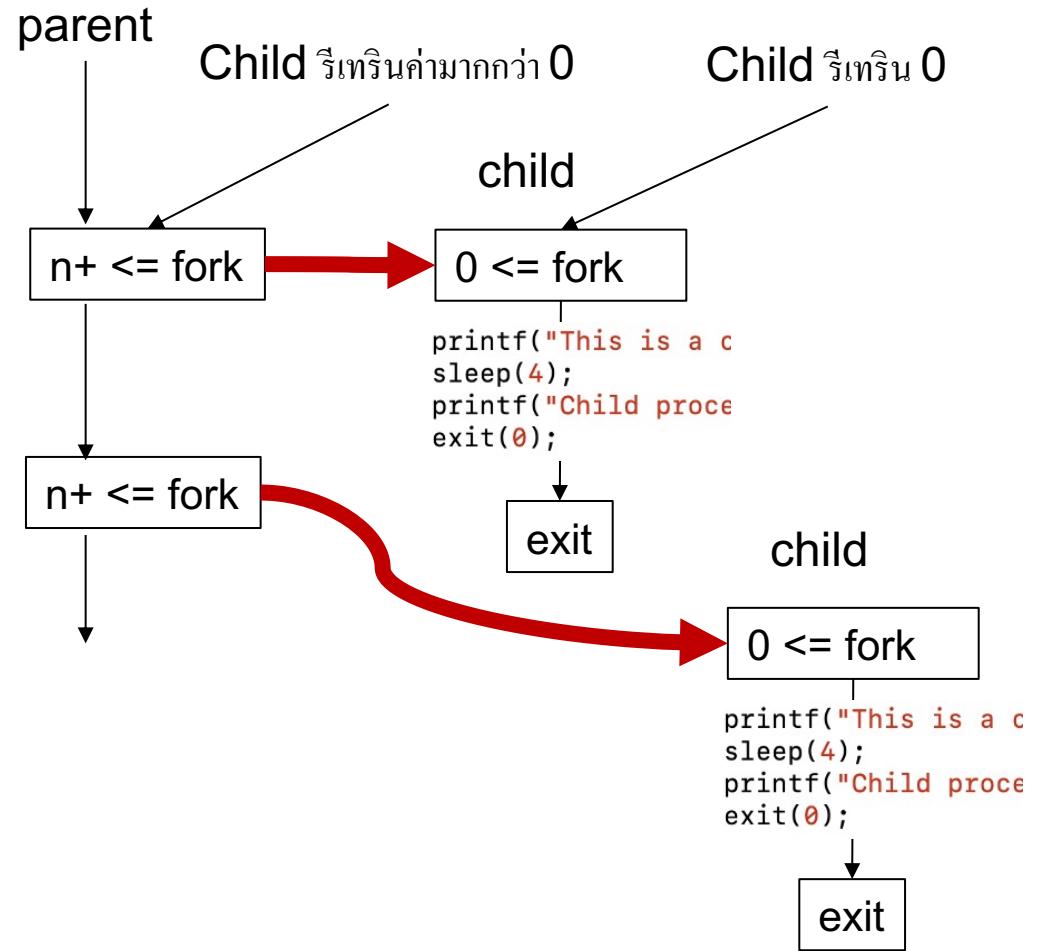


Visualize ex7.c

```
int main(int argc, char * argv[])
{
    pid_t pid1, pid2;
    int status;

    if ((pid1 = fork()) == 0) {
        printf("This is a c
sleep(4);
printf("Child proce
exit(0);
    }
    if ((pid2 = fork()) == 0) {
        printf("This is a c
sleep(4);
printf("Child proce
exit(0);
    }
    /* parent */
    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);

    exit(0);
}
```



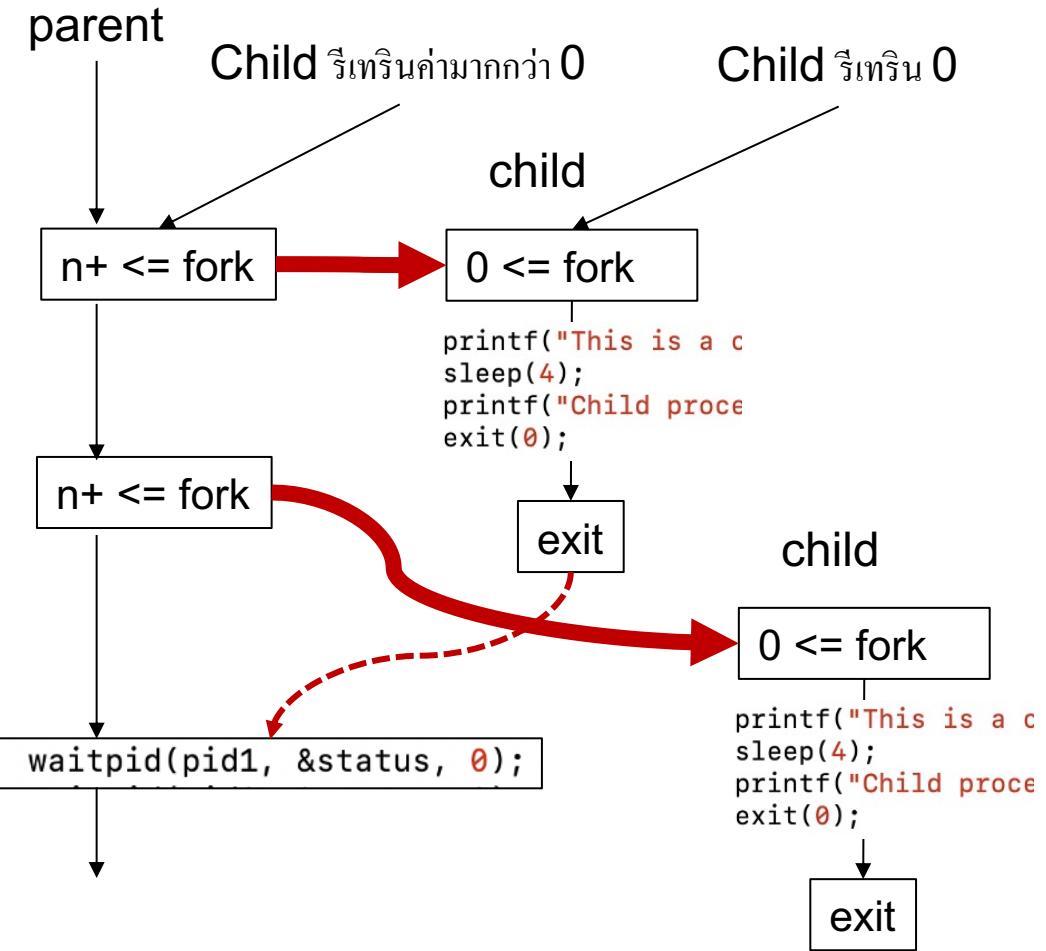


Visualize ex7.c

```
int main(int argc, char * argv[])
{
    pid_t pid1, pid2;
    int status;

    if ((pid1 = fork()) == 0) {
        printf("This is a c");
        sleep(4);
        printf("Child proce");
        exit(0);
    }
    if ((pid2 = fork()) == 0) {
        printf("This is a c");
        sleep(4);
        printf("Child proce");
        exit(0);
    }
    /* parent */
    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);

    exit(0);
}
```



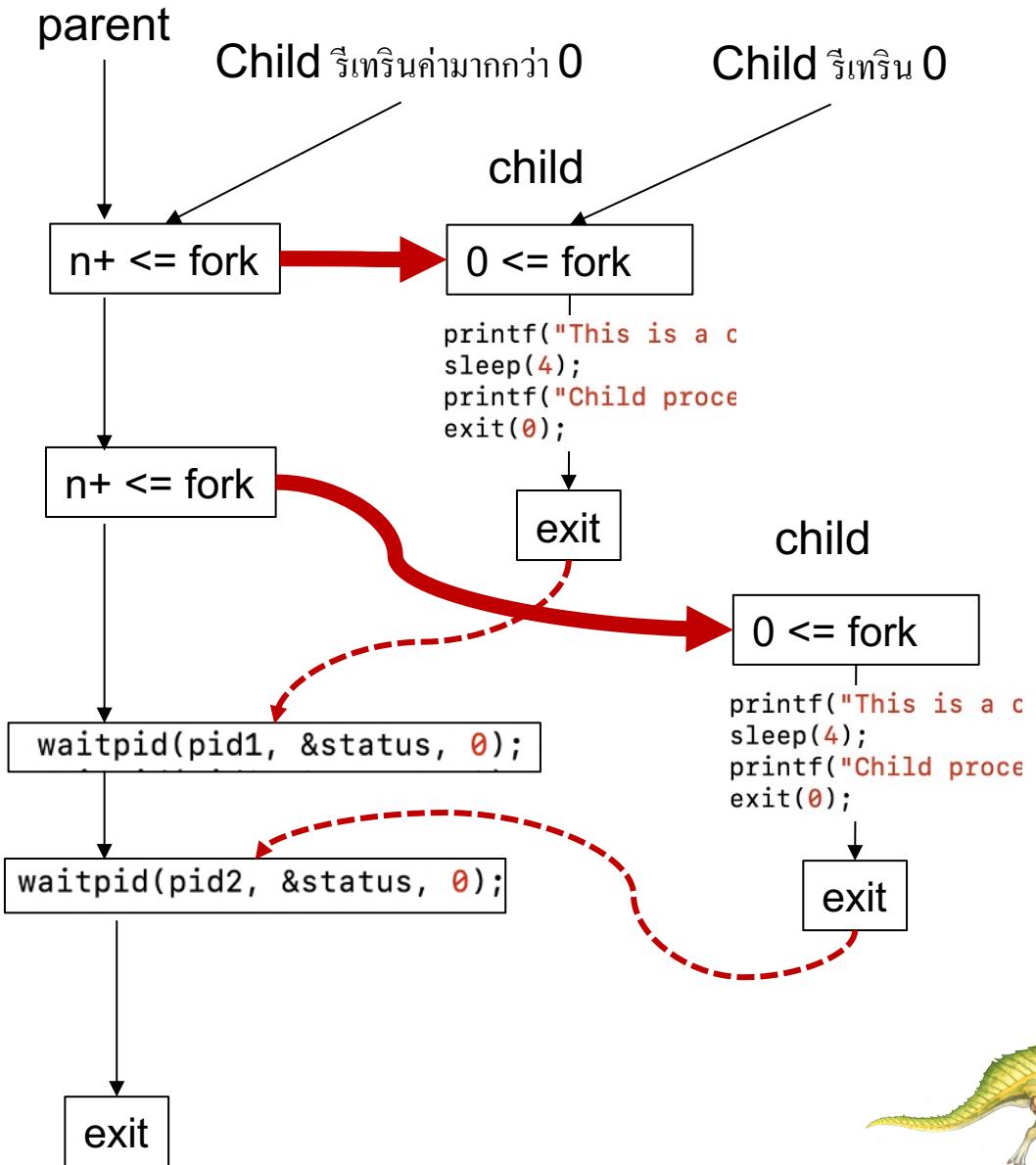


Visualize ex7.c

```
int main(int argc, char * argv[])
{
    pid_t pid1, pid2;
    int status;

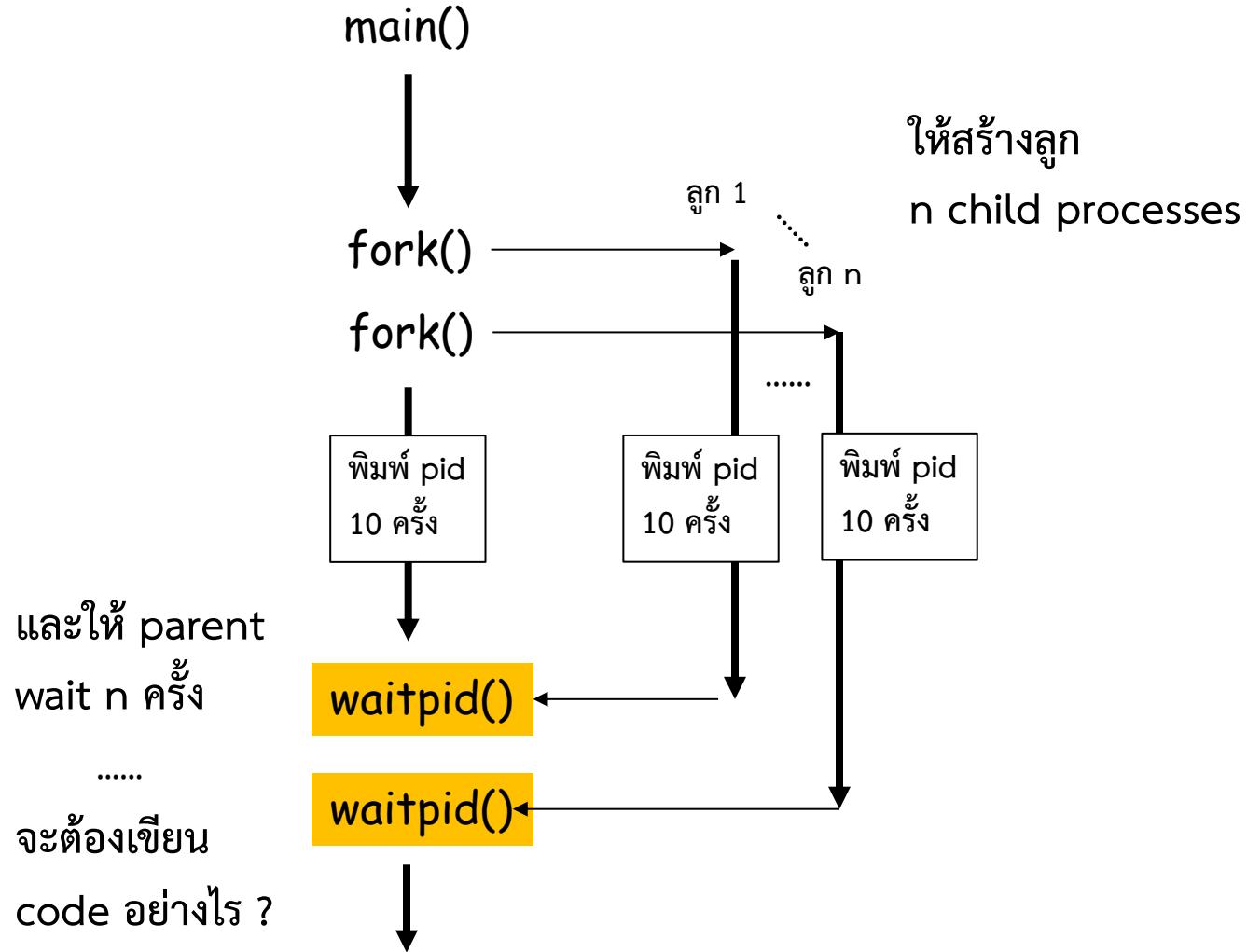
    if ((pid1 = fork()) == 0) {
        printf("This is a c
sleep(4);
printf("Child proce
exit(0);
    }
    if ((pid2 = fork()) == 0) {
        printf("This is a c
sleep(4);
printf("Child proce
exit(0);
    }
    /* parent */
    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);

    exit(0);
}
```





แบบฝึกหัด 1





แปลง ex7.c อย่างไรดี

นศ ต้อง **visualize code** ส่วนนี้แล้วกูร่ำ **fork**

```
int main(int argc, char * argv[])
{
    pid_t pid1, pid2;
    int status;

    if ((pid1 = fork()) == 0)
        printf("This is a child\n");
        sleep(4);
        printf("Child process\n");
        exit(0);
    }
    if ((pid2 = fork()) == 0)
        printf("This is a child\n");
        sleep(4);
        printf("Child process\n");
        exit(0);
    }

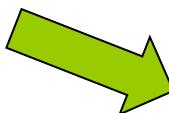
    /* parent */
    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);

    exit(0);
}
```



```
if ((pid1 = fork()) == 0)
    printf("This is a child\n");
    sleep(4);
    printf("Child process\n");
    exit(0);
}
if ((pid2 = fork()) == 0)
    printf("This is a child\n");
    sleep(4);
    printf("Child process\n");
    exit(0);
}
```

}



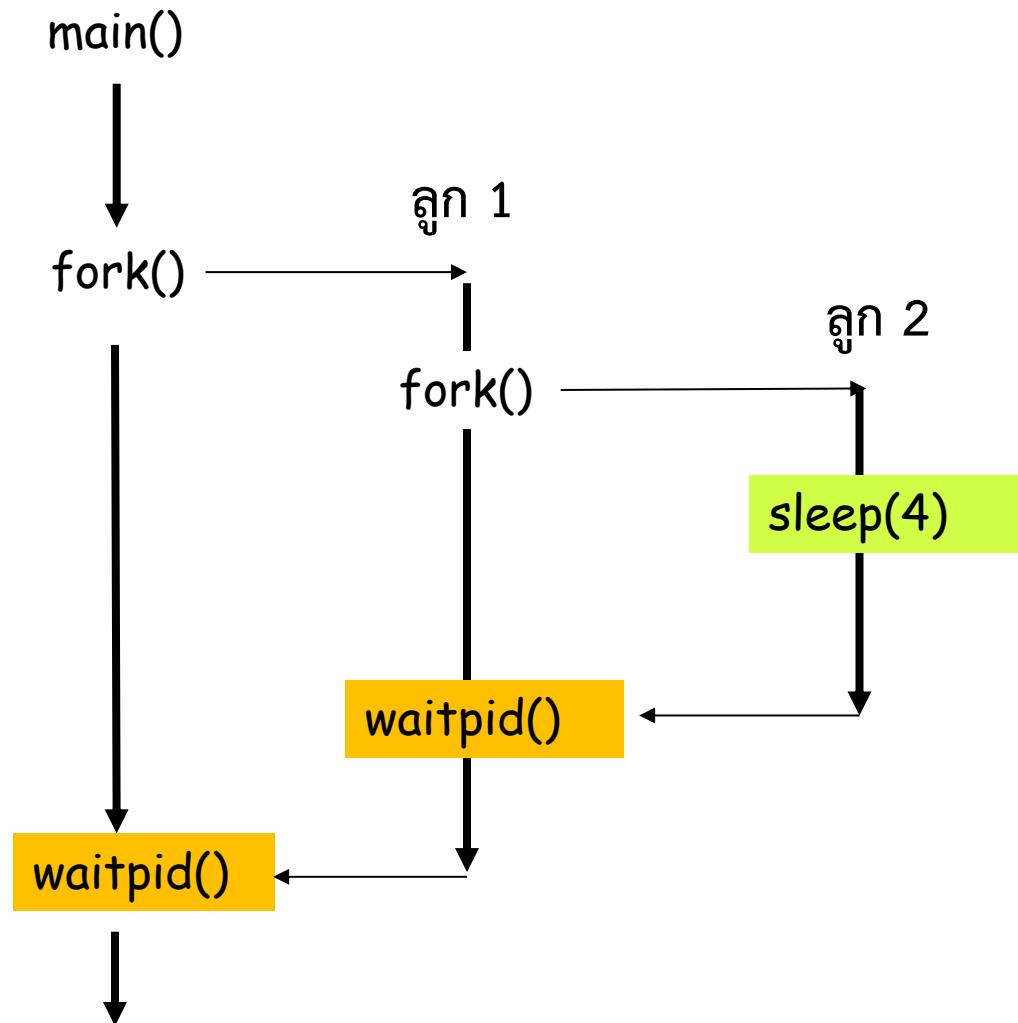
```
/* parent */
waitpid(pid1, &status, 0);
waitpid(pid2, &status, 0);
```

นศ ต้องพิจารณาส่วนนี้แล้วกูร่ำ **parent** รอ (**wait**) ใครมานะ



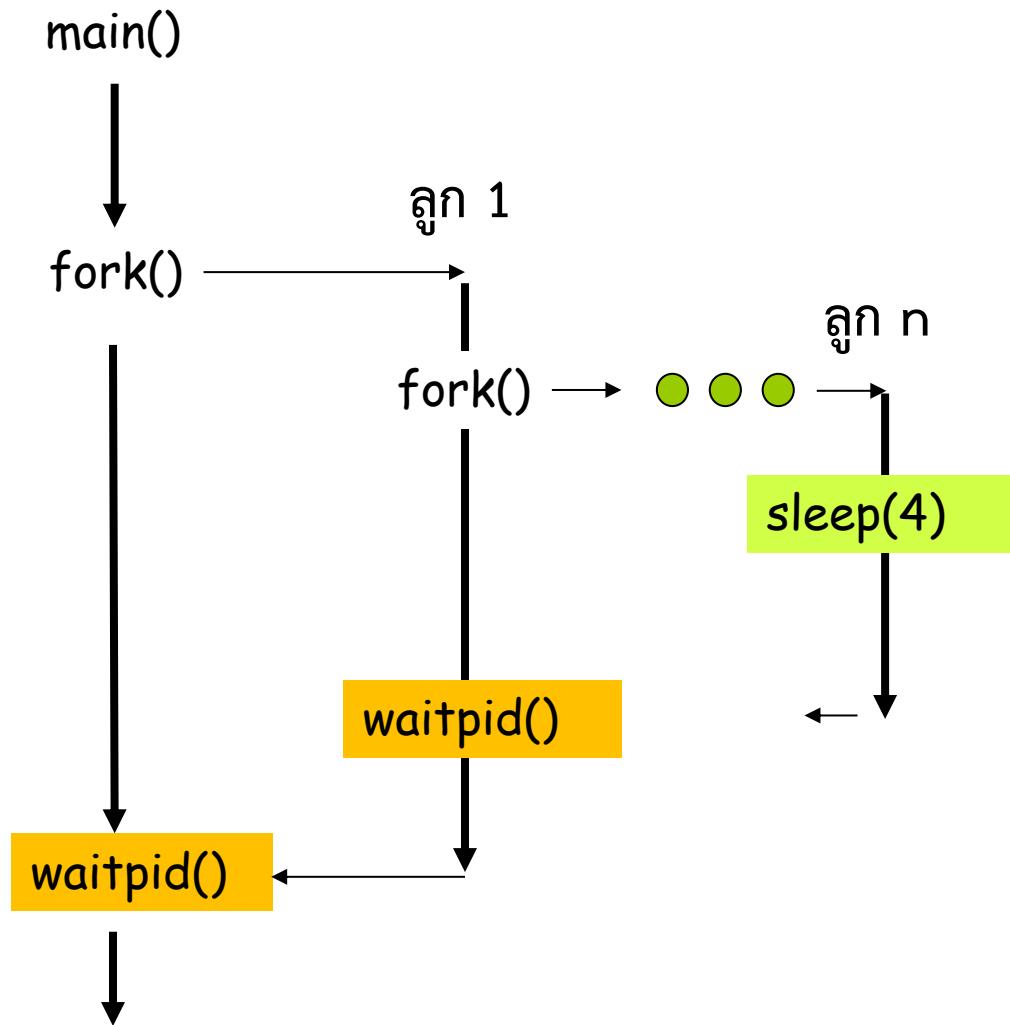


แบบฝึกหัด 2 (ท้าทาย ลองปรึกษา AI ได้)





แบบฝึกหัด 3 (ท้าทาย ลองปรึกษา AI ได้)



ถ้าให้สร้างลูก process ต่อ กันแบบนี้ ก ชั้น จะเขียนโปรแกรม อย่างไร

1. แบบ Recursive
2. แบบ loop





exec system call

1. exec system call เป็นชุดของ Linux system call ที่ใช้สำหรับ load โปรแกรม executable จากหน่วยเก็บข้อมูลขึ้นมาเป็นไฟล์โดยที่ (ถ้า exec ทำงานสำเร็จ) OS จะโหลดเนื้อหาของ Code Data Stack และ Heap ของโปรแกรม executable เข้ามาแทนที่เนื้อหาของไฟล์เดิมในหน่วยความจำและไฟล์จะเริ่มประมวลผลตั้งแต่คำสั่งแรกใน main function ของโปรแกรม executable
2. ถ้า exec รันสำเร็จไฟล์จะรันเนื้อหาของโปรแกรมใหม่
3. แต่ถ้าไม่สำเร็จมันจะ return -1 และ set ค่า error flag status ในตัวแปร global variable errno
4. exec ทำให้โปรแกรมเมอร์สามารถโหลดโปรแกรมใหม่ได้ตอน runtime
5. exec system call มีให้เลือกใช้หลายแบบ ดังนี้





exec

```
#include <unistd.h>

int execl(const char * pathname, const char * arg0, ... /* (char *)0 */ );

int execv(const char * pathname, char *const argv []);
```



```
int execle(const char * pathname, const char * arg0, ...
           /* (char *)0, char *const envp[] */ );
```



```
int execve(const char * pathname, char *const argv[], char *const envp[]);
```



```
int execlp(const char * filename, const char * arg0, ... /* (char *)0 */ );
```



```
int execvp(const char * filename, char *const argv[]);
```



```
int fexecve(int fd, char *const argv[], char *const envp[]);
```





| Function | <i>pathname</i> | <i>filename</i> | <i>fd</i> | Arg list | <i>argv[]</i> | <i>environ</i> | <i>envp[]</i> |
|----------------------|-----------------|-----------------|-----------|----------|----------------|----------------|----------------|
| <code>execl</code> | • | | | • | | • | |
| <code>execlp</code> | | • | | • | | • | |
| <code>execle</code> | • | | | • | | | • |
| <code>execv</code> | • | | | | • | • | |
| <code>execvp</code> | | • | | | • | • | |
| <code>execve</code> | • | | • | | • | | • |
| <code>fexecve</code> | | | | | • | | • |
| (letter in name) | | p | f | l | v | | e |





execlp()

- **execlp()** execute a new command
 - The command must be one of the executables in directories in the \$PATH environments

```
kasidit@enterprise:~$ echo $PATH
/home/kasidit/.local/bin:/usr/local/cuda-12.1/bin:/home/kasidit/.local/bin
:/home/kasidit/Software/qemu/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:
/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
kasidit@enterprise:~$
```

- **execlp()** replaces the caller process with the code of the new command

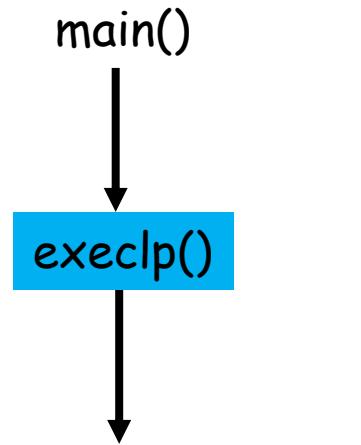




fork, execp, and waitpid

โปรแกรมเดิมรัน

โปรแกรมใหม่เริ่ม
รันจาก main



Load โปรแกรมใหม่เข้ามา
แทนที่ process เดิมและ
(ใช้ pid เดียวกัน)
ปฏิบัติการต่อไป





ex8.c

```
1 #include <stdio.h>
2 #include <signal.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <string.h>
6 #include <sys/wait.h>
7 #include <inttypes.h>
8
9 int main(int argc, char * argv[])
10 {
11     pid_t    pid;
12     int      status;
13
14     pid = getpid();
15     printf("before call exec(), pid = %ld\n", (long)pid); fflush(stdout);
16
17     execlp("/bin/ls", "ls", NULL);
18
19     pid = getpid();
20     printf("after call exec(), pid = %ld\n", (long)pid); fflush(stdout);
21     exit(0);
22 }
```

~





```
1 #include <stdio.h>
2 #include <signal.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <string.h>
6 #include <sys/wait.h>
7 #include <inttypes.h>
8
9 int main(int argc, char * argv[])
10 {
11     pid_t    pid;
12     int      status;
13
14     pid = getpid();
15     printf("before call exec(), pid = %ld\n", (long)pid); fflush(stdout);
16
17     execlp("/bin/ls", "ls", NULL);
18
19     pid = getpid();
20     printf("after call exec(), pid = %ld\n", (long)pid); fflush(stdout);
21     exit(0);
22 }
```

~

```
u2909610384@cs22201:~/ch3/Solutions$ gcc -o exc exec1.c
u2909610384@cs22201:~/ch3/Solutions$
u2909610384@cs22201:~/ch3/Solutions$
u2909610384@cs22201:~/ch3/Solutions$
u2909610384@cs22201:~/ch3/Solutions$ ./exc
before call exec(), pid = 221825
exc exec1.c fe fk forkexec1.c forkexec1.c nozombie.c nz output.txt z1 z2 zombie1.c zombie2.c
u2909610384@cs22201:~/ch3/Solutions$
u2909610384@cs22201:~/ch3/Solutions$
```





ex9.c

```
kasidit@enterprise:~/Desktop/CS435/22024$ export PATH=$PATH:.  
kasidit@enterprise:~/Desktop/CS435/22024$ echo $PATH  
/home/kasidit/.local/bin:/usr/local/cuda-12.1/bin:/home/kasidit/.local/bin:/home/kasidi  
usr/local/bin:/usr/sbin:/usr/bin:/sbin:/usr/games:/usr/local/games:/snap/bin:...  
...
```

```
1 #include <stdio.h>  
2 #include <unistd.h>  
3 #include <errno.h>  
4  
5 int main() {  
6  
7     printf("Before execlp()\n");  
8  
9     execlp("ex9", "ex9", NULL, NULL);  
10  
11    printf("After execlp()\n");  
12    return 0;  
13 }
```





ex9.c

main()

โปรแกรมเดิม

execlp()

โปรแกรมใหม่

execlp()

โปรแกรมใหม่

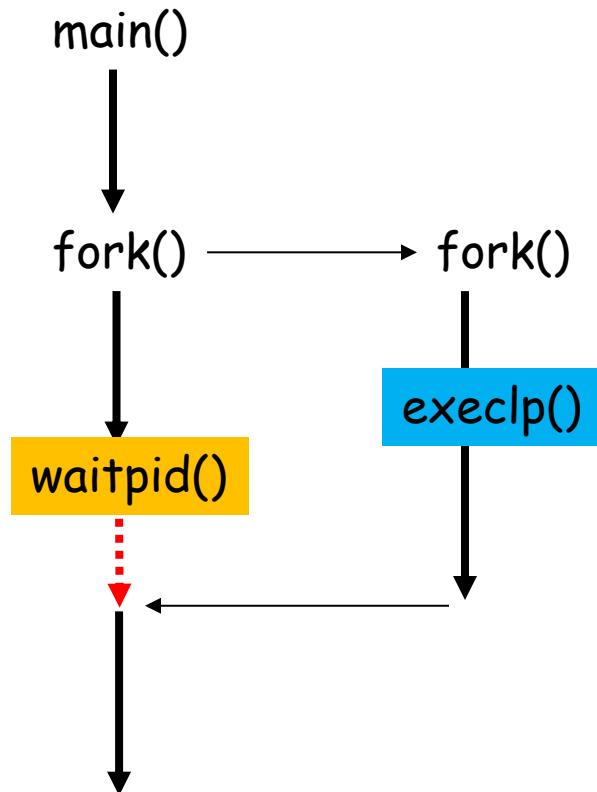
Load โปรแกรมใหม่เข้ามา
แทนที่ process และ
เริ่มรัน

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <errno.h>
4
5 int main() {
6
7     printf("Before execlp()\n");
8
9     execlp("ex9", "ex9", NULL, NULL);
10
11    printf("After execlp()\n");
12
13 }
```

Before execlp()
Before execlp()



fork, execp, and waitpid



Load โปรแกรมใหม่เข้ามา
แทนที่ child process และ
ปฏิบัติการต่อไป

รันจนจบการทำงาน

C Program Forking Separate Process

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait(NULL);
        printf("Child Complete");
    }
}

return 0;
}
```





ex10.c

```
8
9 int main(int argc, char * argv[])
10 {
11     pid_t    pid;
12     pid_t    mypid;
13     int      status;
14
15     if ((pid = fork()) < 0) {
16         exit(1);
17     }
18     else if (pid == 0) {
19         mypid = getpid();
20         printf("child pid = %ld\n", (long)mypid); fflush(stdout);
21         //
22         execlp("/bin/ls", "ls", NULL);
23         exit(1);
24     }
25     else{
26         mypid = getpid();
27         printf("parent pid = %ld\n", (long)mypid); fflush(stdout);
28         waitpid(pid, &status, 0);
29         exit(0);
30     }
31     exit(0);
32 }
```







shell1.c

```
1 /* This code is modified from examples in the Advanced Unix Programming book by R
2 #include <stdio.h>
3 #include <signal.h>
4 #include <stdlib.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include <sys/wait.h>
8 #include <inttypes.h>
9
10#define MAXLINE 500
11#define MAXARGS 50
12
13int
14main(void)
15{
16    char    buf[MAXLINE];
17    pid_t   pid;
18    int     status;
19    int     i, sargsnum;
20
21    char    *sargs[MAXARGS];
22    char    *p;
23
24    printf("%% "); /* print prompt (printf requires %% to print %) */
25    while (fgets(buf, MAXLINE, stdin) != NULL) {
26        if (buf[strlen(buf) - 1] == '\n')
27            buf[strlen(buf) - 1] = 0; /* replace newline with null */
28    }
```





```
28
29         if ((pid = fork()) < 0) {
30             printf("fork error");
31             exit(1);
32         } else if (pid == 0) { /* child */
33             execlp(buf, buf, (char *)0);
34             printf("couldn't execute: %s", sargs[0]);
35             exit(127);
36         }
37
38         /* parent */
39         if ((pid = waitpid(pid, &status, 0)) < 0)
40             printf("waitpid error");
41         printf("%% ");
42     }
43     exit(0);
44 }
```





Main loop

```
24     printf("%% "); /* print prompt (printf requires %% to print %) */
25     while (fgets(buf, MAXLINE, stdin) != NULL) {
26         if (buf[strlen(buf) - 1] == '\n')
27             buf[strlen(buf) - 1] = 0; /* replace newline with null */
28
29         if ((pid = fork()) < 0) {
30             printf("fork error");
31             exit(1);
32         } else if (pid == 0) {           /* child */
33             execlp(buf, buf, (char *)0);
34             printf("couldn't execute: %s", sargs[0]);
35             exit(127);
36         }
37
38         /* parent */
39         if ((pid = waitpid(pid, &status, 0)) < 0)
40             printf("waitpid error");
41         printf("%% ");
42     }
```





Compile and run program

```
gcc -o s1 shell1.c
```

```
[kani@Kasidits-MacBook-Pro 2-shell12 % ./s1
% date
Wed Mar 13 14:22:22 +07 2024
% ls
Makefile           insercomment.sh s1          shell1.c
% ps
  PID TTY          TIME CMD
16468 ttys000    0:00.01 ./s1
60565 ttys000    0:00.83 -zsh
% who
kani      console  Feb 28 09:47
kani      ttys000  Mar   5 09:14
% ^C
[kani@Kasidits-MacBook-Pro 2-shell12 %
kani@Kasidits-MacBook-Pro 2-shell12 %
```



สรุป

- Process State Transition Diagram
- Linux Process State
- ตัวอย่างโปรแกรม
- แบบฝึกหัด