

CSGE602055 Operating Systems

CSF2600505 Sistem Operasi

Week 06: Concurrency: Processes & Threads

Rahmat M. Samik-Ibrahim (ed.)

University of Indonesia

<https://os.vlsm.org/>

Always check for the latest revision!

REV252 12-Oct-2020

Operating Systems 202³) — PJJ from HOME

ZOOM: International [Tue 08-10] — A/Matrix [Tue 10-12]

Week	Schedule & Deadline ¹⁾	Topic	OSC10 ²⁾
Week 00	15 Sep - 21 Sep 2020	Overview 1, Virtualization & Scripting	Ch. 1, 2, 18.
Week 01	22 Sep - 28 Sep 2020	Overview 2, Virtualization & Scripting	Ch. 1, 2, 18.
Week 02	29 Sep - 05 Oct 2020	Security, Protection, Privacy, & C-language.	Ch. 16, 17.
Week 03	06 Oct - 12 Oct 2020	File System & FUSE	Ch. 13, 14, 15.
Week 04	13 Oct - 19 Oct 2020	Addressing, Shared Lib, & Pointer	Ch. 9.
Week 05	20 Oct - 26 Oct 2020	Virtual Memory	Ch. 10.
Week 06	27 Oct - 16 Nov 2020 29 Oct 2020	Concurrency: Processes & Threads Maulid Nabi	Ch. 3, 4.
Week 07	17 Nov - 23 Nov 2020	Synchronization & Deadlock	Ch. 6, 7, 8.
Week 08	24 Nov - 30 Nov 2020	Scheduling + W06/W07	Ch. 5.
Week 09	01 Dec - 07 Dec 2020	Storage, Firmware, Bootloader, & Systemd	Ch. 11.
Week 10	08 Dec - 16 Dec 2020	I/O & Programming	Ch. 12.
	09 Dec 2020	Pil Kada	

¹⁾ The **DEADLINE** of Week 00 is 21 Sep 2020, whereas the **DEADLINE** of Week 01 is 28 Sep 2020, and so on...

²⁾ Silberschatz et. al.: **Operating System Concepts**, 10th Edition, 2018.

³⁾ This information will be on **EVERY** page two (2) of this course material.

STARTING POINT — <https://os.vlsm.org/>

- **Text Book** — Any recent/decent OS book. Eg. (**OSC10**) Silberschatz et. al.: **Operating System Concepts**, 10th Edition, 2018. See also <http://codex.cs.yale.edu/avi/os-book/OS10/>.
- **Resources**
 - **SCELE** — <https://scele.cs.ui.ac.id/course/view.php?id=3020>. The enrollment key is **XXX**.
 - **Download Slides and Demos from GitHub.com**
<https://github.com/UI-FASILKOM-OS/SistemOperasi/>:
os00.pdf (W00), os01.pdf (W01), os02.pdf (W02), os03.pdf (W03),
os04.pdf (W04), os05.pdf (W05), os06.pdf (W06), os07.pdf (W07),
os08.pdf (W08), os09.pdf (W09), os10.pdf (W10).
 - **Problems** — <https://rms46.vlsm.org/2/>:
195.pdf (W00), 196.pdf (W01), 197.pdf (W02), 198.pdf (W03),
199.pdf (W04), 200.pdf (W05), 201.pdf (W06), 202.pdf (W07),
203.pdf (W08), 204.pdf (W09), 205.pdf (W10).
- **Build your own Virtual Guest**
<https://osp4diss.vlsm.org/>

Agenda I

- 1 Start
- 2 Schedule
- 3 Agenda
- 4 Week 06
- 5 Week 06
- 6 Process Map
- 7 Process State
- 8 Makefile
- 9 00-show-pid
- 10 01-fork
- 11 02-fork
- 12 03-fork
- 13 01-fork vs 02-fork vs 03-fork
- 14 04-sleep
- 15 05-fork
- 16 06-fork

Agenda II

- 17 07-execlp
- 18 08-fork
- 19 09-fork
- 20 10-fork
- 21 11-fork
- 22 12-fork
- 23 13-uas161
- 24 14-uas162
- 25 15-uas171
- 26 16-uas172
- 27 Week 06: Check List
- 28 The End

Week 06 Concurrency: Topics¹

- States and state diagrams
- Structures (ready list, process control blocks, and so forth)
- Dispatching and context switching
- The role of interrupts
- Managing atomic access to OS objects
- Implementing synchronization primitives
- Multiprocessor issues (spin-locks, reentrancy)

¹Source: ACM IEEE CS Curricula 2013

Week 06 Concurrency: Learning Outcomes (1)¹

- Describe the need for concurrency within the framework of an operating system. [Familiarity]
- Demonstrate the potential run-time problems arising from the concurrent operation of many separate tasks. [Usage]
- Summarize the range of mechanisms that can be employed at the operating system level to realize concurrent systems and describe the benefits of each. [Familiarity]
- Explain the different states that a task may pass through and the data structures needed to support the management of many tasks. [Familiarity]

¹Source: ACM IEEE CS Curricula 2013

Week 06 Concurrency: Learning Outcomes (2)¹

- Summarize techniques for achieving synchronization in an operating system (e.g., describe how to implement a semaphore using OS primitives). [Familiarity]
- Describe reasons for using interrupts, dispatching, and context switching to support concurrency in an operating system. [Familiarity]
- Create state and transition diagrams for simple problem domains. [Usage]

¹Source: ACM IEEE CS Curricula 2013

Week 06: Concurrency: Processes & Threads

- Reference: (OSC10-ch03 OSC10-ch04 demo-w06)
- Process Concept
 - Program (passive) \leftrightarrow Process (active)
 - Process in Memory: | *Stack* \cdots *Heap* | *Data* | *Text* |
 - Process State: | *running* | *waiting* | *ready* |
 - Process Control Block (PCB)
 - /proc/, Process State, Program Counter, Registers, Management Information.
- Process Creation
 - PID: Process Identifier (uniq)
 - The Parent Process forms a tree of Children Processes
 - `fork()`, new process system call (clone)
 - `exec1p()`, replaces the clone with a new program.
- Process Termination
 - `wait()`, until the child process is terminated.
- PCB (Context) Switch

A PROCESS IN MEMORY



(c) 2017 VauLSMorg

Figure: A Process in (**logical**) Memory

Process State

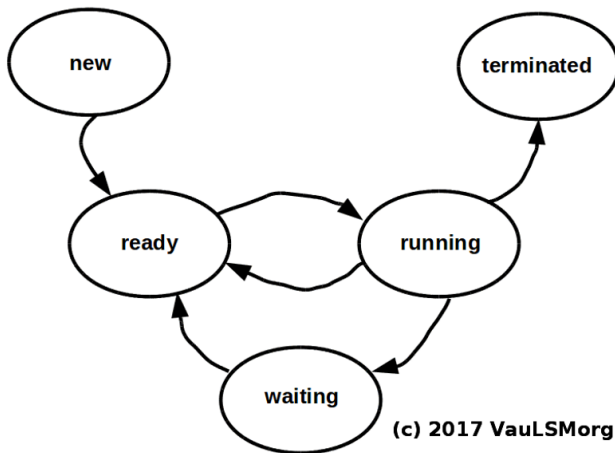


Figure: A Process State

Process Scheduling

- Scheduling Queue
- Schedulers
 - Long Term (non VM) vs Short Term (CPU)
 - (I/O vs CPU) Bound Processes
- Context Switch
- I/O Queue Scheduling
- Android Systems
 - Dalvik VM Performance Problem: Replaced with ART (Android Runtime).
 - Foreground Processes: with an User Interface (UI) for Videos, Images, Sounds, Texts, etc.
 - Background Processes: with a service with no UI and small memory footprint.

Inter-Process Communication (IPC)

- Independent vs Cooperating Processes.
 - Cooperation: Information Sharing, Computational Speedup, Modularity, Convenience.
- Shared Memory vs Message Passing.
 - Message Passing: Direct vs Indirect Communication
- Client-Server Systems
 - Sockets
 - RPC: Remote Procedure Calls
 - Pipes

- Single vs Multithreaded Process
 - MultiT Benefits: Responsiveness, Resource Sharing, Economy, Scalability
- Multicore Programming
 - Concurrency vs. Parallelism
- Multithreading Models (Kernel vs User Thread)
 - Many to One
 - One to One
 - Many to Many
 - Multilevel Models
- Threading Issues
 - Parallelism on a multi-core system.
- Pthreads

Makefile

```
CC='gcc'
CFLAGS='-std=c99'

P00=00-show-pid
...
P15=15-uas171
P16=16-uas172

EXECS= \
    $(P00) \
    $(P01) \
    ...
    $(P15) \
    $(P16) \

all: $(EXECS)

$(P00): $(P00).c
    $(CC) $(P00).c -o $(P00)

$(P01): $(P01).c
    $(CC) $(P01).c -o $(P01)
...

$(P15): $(P15).c
    $(CC) $(P15).c -o $(P15)

$(P16): $(P16).c
    $(CC) $(P16).c -o $(P16)

clean:
    rm -f $(EXECS)
```

00-show-pid

```
/*
 * (c) 2016-2020 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 * REV07 Tue Mar 24 12:06:10 WIB 2020
 * START Mon Oct 24 09:42:05 WIB 2016
 */

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

void main(void) {
    printf("  [[[ This is 00-show-pid: PID[%d] PPID[%d] ]]]\n",
           getpid(), getppid());
}

>>>> $ ./00-show-pid

[[[ This is 00-show-pid: PID[5777] PPID[1350] ]]]
```


01-fork

```
>>>> $ cat 01-fork.c ; echo "=====" ; ./01-fork
/* (c) 2016-2017 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 */

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

void main(void) {
    char *iAM="PARENT";

    printf("PID[%d] PPID[%d] (START:%s)\n", getpid(), getppid(), iAM);
    if (fork() > 0) {
        sleep(1);          /* LOOK THIS ***** */
        printf("PID[%d] PPID[%d] (IFFO:%s)\n", getpid(), getppid(), iAM);
    } else {
        iAM="CHILD";
        printf("PID[%d] PPID[%d] (ELSE:%s)\n", getpid(), getppid(), iAM);
    }
    printf("PID[%d] PPID[%d] (STOP:%s)\n", getpid(), getppid(), iAM);
}

=====
PID[5784] PPID[1350] (START:PARENT)
PID[5785] PPID[5784] (ELSE:CHILD)
PID[5785] PPID[5784] (STOP:CHILD)
PID[5784] PPID[1350] (IFFO:PARENT)
PID[5784] PPID[1350] (STOP:PARENT)
>>>> $
```

02-fork

```
>>>>> $ cat 02-fork.c ; echo "=====" ; ./02-fork
/* (c) 2016-2017 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 */

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

void main(void) {
    char *iAM="PARENT";

    printf("PID[%d] PPID[%d] (START:%s)\n", getpid(), getppid(), iAM);
    if (fork() > 0) {
        printf("PID[%d] PPID[%d] (IFFO:%s)\n", getpid(), getppid(), iAM);
    } else {
        iAM="CHILD";
        printf("PID[%d] PPID[%d] (ELSE:%s)\n", getpid(), getppid(), iAM);
        sleep(1);      /* LOOK THIS ***** */
    }
    printf("PID[%d] PPID[%d] (STOP:%s)\n", getpid(), getppid(), iAM);
}

=====
PID[5792] PPID[1350] (START:PARENT)
PID[5792] PPID[1350] (IFFO:PARENT)
PID[5792] PPID[1350] (STOP:PARENT)
PID[5793] PPID[5792] (ELSE:CHILD)
>>>>> $ PID[5793] PPID[1] (STOP:CHILD)
>>>>> $
```

03-fork

```
>>>> $ cat 03-fork.c ; echo "=====" ; ./03-fork
/* (c) 2016-2017 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 */

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

void main(void) {
    char *iAM="PARENT";

    printf("PID[%d] PPID[%d] (START:%s)\n", getpid(), getppid(), iAM);
    if (fork() > 0) {
        wait(NULL);          /* LOOK THIS ***** */
        printf("PID[%d] PPID[%d] (IFFO:%s)\n", getpid(), getppid(), iAM);
    } else {
        iAM="CHILD";
        printf("PID[%d] PPID[%d] (ELSE:%s)\n", getpid(), getppid(), iAM);
    }
    printf("PID[%d] PPID[%d] (STOP:%s)\n", getpid(), getppid(), iAM);
}

=====
PID[5799] PPID[1350] (START:PARENT)
PID[5800] PPID[5799] (ELSE:CHILD)
PID[5800] PPID[5799] (STOP:CHILD)
PID[5799] PPID[1350] (IFFO:PARENT)
PID[5799] PPID[1350] (STOP:PARENT)
>>>> $
```

01-fork vs 02-fork vs 03-fork

```
>>>>> $ ./01-fork
PID[5803] PPID[1350] (START:PARENT)
PID[5804] PPID[5803] (ELSE:CHILD)
PID[5804] PPID[5803] (STOP:CHILD)
PID[5803] PPID[1350] (IFF0:PARENT)
PID[5803] PPID[1350] (STOP:PARENT)
>>>>> $ ./02-fork
PID[5805] PPID[1350] (START:PARENT)
PID[5805] PPID[1350] (IFF0:PARENT)
PID[5805] PPID[1350] (STOP:PARENT)
PID[5806] PPID[5805] (ELSE:CHILD)
>>>>> $ PID[5806] PPID[1] (STOP:CHILD)

>>>>> $ ./03-fork
PID[5807] PPID[1350] (START:PARENT)
PID[5808] PPID[5807] (ELSE:CHILD)
PID[5808] PPID[5807] (STOP:CHILD)
PID[5807] PPID[1350] (IFF0:PARENT)
PID[5807] PPID[1350] (STOP:PARENT)
>>>>> $
```

04-sleep

```
#include <stdio.h>
#include <unistd.h>
void main(void) {
    int ii;
    printf("Sleeping 3s with fflush(): ");
    fflush(NULL);
    for (ii=0; ii < 3; ii++) {
        sleep(1);
        printf("x ");
        fflush(NULL);
    }
    printf("\nSleeping with no fflush(): ");
    for (ii=0; ii < 3; ii++) {
        sleep(1);
        printf("x ");
    }
    printf("\n");
}
Sleeping 3s with fflush(): x x x
Sleeping with no fflush(): x x x
```

```

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

void main(void) {
    printf("Start:          PID[%d] PPID[%d]\n", getpid(), getppid());
    fflush(NULL);
    if (fork() == 0) {
        /* START BLOCK
        execlp("./00-fork", "00-fork", NULL);
        END BLOCK */
        printf("Child:          ");
    } else {
        wait(NULL);
        printf("Parent:          ");
    }
    printf("          PID[%d] PPID[%d] <<< <<< <<<\n", getpid(), getppid());
}

```

no execlp =====

```

Start:          PID[6040] PPID[1350]
Child:          PID[6041] PPID[6040] <<< <<< <<<
Parent:         PID[6040] PPID[1350] <<< <<< <<<

```

05b-fork

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

void main(void) {
    printf("Start:          PID[%d] PPID[%d]\n", getpid(), getppid());
    fflush(NULL);
    if (fork() == 0) {
        /* START BLOCK
           END   BLOCK */
        execlp("./00-fork", "00-fork", NULL);
        printf("Child:          ");
    } else {
        wait(NULL);
        printf("Parent:          ");
    }
    printf("          PID[%d] PPID[%d] <<< <<< <<<\n", getpid(), getppid());
}

execlp =====
Start:          PID[6007] PPID[1350]
[[[ This is 00-show-pid: PID[6008] PPID[6007] ]]]
Parent:          PID[6007] PPID[1350] <<< <<< <<<
```

06a-fork

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
/***** main ***/
void main(void) {
    pid_t val1, val2, val3;
    val3 = val2 = val1 = 1000;
    printf("PID==%4d ==== \n", getpid());
    fflush(NULL);
    val1 = fork();
    wait(NULL);
    val2 = fork();
    wait(NULL);
    val3 = fork();
    wait(NULL);
    /* ***** START BLOCK *
       ***** END** BLOCK */
    printf("VAL1=%4d VAL2=%4d VAL3=%4d\n", val1, val2, val3);
}

=====
PID==[13965] ====
VAL1=[01000] VAL2=[01000] VAL3=[01000]
```



```

#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
/***** main ** */
void main(void) {
    pid_t val1, val2, val3;
    val3 = val2 = val1 = 1000;
    printf("PID==%4d ==== \n", getpid());
    fflush(NULL);
    val1 = fork();
    wait(NULL);
/* ***** START BLOCK */
    val2 = fork();
    wait(NULL);
    val3 = fork();
    wait(NULL);
    ***** END** BLOCK */
    printf("VAL1=%4d VAL2=%4d VAL3=%4d\n", val1, val2, val3);
}

=====
PID==[13969] ==== 
VAL1=[00000] VAL2=[01000] VAL3=[01000]
VAL1=[13970] VAL2=[01000] VAL3=[01000]

```

```

#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
/***** main ***/
void main(void) {
    pid_t val1, val2, val3;
    val3 = val2 = val1 = 1000;
    printf("PID==%4d ==== \n", getpid());
    fflush(NULL);
    val1 = fork();
    wait(NULL);
    val2 = fork();
    wait(NULL);
    /* ***** START BLOCK */
    val3 = fork();
    wait(NULL);
    /* ***** END** BLOCK */
    printf("VAL1=%4d VAL2=%4d VAL3=%4d\n", val1, val2, val3);
}

=====
PID==[13971] ==== 
VAL1=[00000] VAL2=[00000] VAL3=[01000]
VAL1=[00000] VAL2=[13973] VAL3=[01000]
VAL1=[13972] VAL2=[00000] VAL3=[01000]
VAL1=[13972] VAL2=[13974] VAL3=[01000]

```

06d-fork

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
/***** main ***/
void main(void) {
    pid_t val1, val2, val3;
    val3 = val2 = val1 = 1000;
    printf("PID==%4d ==== \n", getpid());
    fflush(NULL);
    val1 = fork();
    wait(NULL);
    val2 = fork();
    wait(NULL);
    val3 = fork();
    wait(NULL);
    /* ***** START BLOCK *
       ***** END** BLOCK */
    printf("VAL1=%4d VAL2=%4d VAL3=%4d\n", val1, val2, val3);
}

=====
PID==[13976] ====
VAL1=[00000] VAL2=[00000] VAL3=[00000]
VAL1=[00000] VAL2=[00000] VAL3=[13979]
VAL1=[00000] VAL2=[13978] VAL3=[00000]
VAL1=[00000] VAL2=[13978] VAL3=[13980]
VAL1=[13977] VAL2=[00000] VAL3=[00000]
VAL1=[13977] VAL2=[00000] VAL3=[13982]
VAL1=[13977] VAL2=[13981] VAL3=[00000]
VAL1=[13977] VAL2=[13981] VAL3=[13983]
```

07-execlp

```
>>>> $ cat 07-execlp.c
/* (c) 2019-2020 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 * REV01 Tue Mar 24 16:29:50 WIB 2020
 * START Mon Dec 9 16:28:36 WIB 2019
 */

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

void main(int argc, char* argv[]) {
    printf("START %11s PID[%d]\n", argv[0], getpid());
    if(argc == 1) {
        execlp(argv[0], "EXECLP", "WhatEver", NULL);
    } else {
        printf("ELSE %11s PID[%d]\n", argv[1], getpid());
    }
    printf("END %11s PID[%d]\n", argv[0], getpid());
}

$ ./07-execlp
START ./07-execlp PID[14172]
START EXECLP PID[14172]
ELSE WhatEver PID[14172]
END EXECLP PID[14172]
$ ./07-execlp XYZZYPLUGH
START ./07-execlp PID[14174]
ELSE XYZZYPLUGH PID[14174]
END ./07-execlp PID[14174]
$
```

08-fork

```
/* (c) 2005-2017 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 * REV02 Thu Oct 26 12:27:30 WIB 2017
 * START 2005
 */
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void main(void) {
    int ii=0;
    if (fork() == 0) ii++;
    wait(NULL);
    if (fork() == 0) ii++;
    wait(NULL);
    if (fork() == 0) ii++;
    wait(NULL);
    printf ("Result = %d \n",ii);
    exit(0);
}
=====
Result = 3
Result = 2
Result = 2
Result = 1
Result = 2
Result = 1
Result = 1
Result = 0
>>>>> $
```

```
/*
 * (c) 2015-2017 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * REV03 Mon Oct 30 11:04:10 WIB 2017
 * REV00 Mon Oct 24 10:43:00 WIB 2016
 * START 2015
 */

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

void main(void) {
    int value;

    value=fork();
    wait(NULL);
    printf("I am PID[%4d] -- The fork() return value is: %4d\n", getpid(), value);

    value=fork();
    wait(NULL);
    printf("I am PID[%4d] -- The fork() return value is: %4d\n", getpid(), value);
}

=====
I am PID[6225] -- The fork() return value is:    0)
I am PID[6226] -- The fork() return value is:    0)
I am PID[6225] -- The fork() return value is: 6226)
I am PID[6224] -- The fork() return value is: 6225)
I am PID[6227] -- The fork() return value is:    0)
I am PID[6224] -- The fork() return value is: 6227)
>>>>> $
```

10-fork

```
/* (c) 2016-2017 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 * REV02 Mon Oct 30 20:25:44 WIB 2017
 */

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

void procStatus(int level) {
    printf("L%d: PID[%d] (PPID[%d])\n", level, getpid(), getppid());
    fflush(NULL);
}

int addLevelAndFork(int level) {
    if (fork() == 0) level++;
    wait(NULL);
    return level;
}

void main(void) {
    int level = 0;
    procStatus(level);
    level = addLevelAndFork(level);
    procStatus(level);
}

=====
L0: PID[7540] (PPID[1350])
L1: PID[7541] (PPID[7540])
L0: PID[7540] (PPID[1350])
```

11-fork

```
/* (c) 2016-2017 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 * REV02 Mon Oct 30 20:27:24 WIB 2017
 * START Mon Oct 24 09:42:05 WIB 2016
 */

#define LOOP 3
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

void procStatus(int level) {
    printf("L%d: PID[%d] (PPID[%d])\n", level, getpid(), getppid());
    fflush(NULL);
}

int addLevelAndFork(int level) {
    if (fork() == 0) level++;
    wait(NULL);
    return level;
}

void main(void) {
    int ii, level = 0;
    procStatus(level);
    for (ii=0;ii<LOOP;ii++) {
        level = addLevelAndFork(level);
        procStatus(level);
    }
}
```


11-fork (2)

```
L0: PID[7548] (PPID[1350])
L1: PID[7549] (PPID[7548])
L2: PID[7550] (PPID[7549])
L3: PID[7551] (PPID[7550])
L2: PID[7550] (PPID[7549])
L1: PID[7549] (PPID[7548])
L2: PID[7552] (PPID[7549])
L1: PID[7549] (PPID[7548])
L0: PID[7548] (PPID[1350])
L1: PID[7553] (PPID[7548])
L2: PID[7554] (PPID[7553])
L1: PID[7553] (PPID[7548])
L0: PID[7548] (PPID[1350])
L1: PID[7555] (PPID[7548])
L0: PID[7548] (PPID[1350])
```

12-fork

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

void waitAndPrintPID(void) {
    wait(NULL);
    printf("PID: %d\n", getpid());
    fflush(NULL);
}

void main(int argc, char *argv[]) {
    int rc, status;

    waitAndPrintPID();
    rc = fork();
    waitAndPrintPID();
    if (rc == 0) {
        fork();
        waitAndPrintPID();
        execlp("./00-fork", "00-fork", NULL);
    }
    waitAndPrintPID();
}

=====
PID: 7614
PID: 7615
PID: 7616
[[[ This is 00-fork: PID[7616] PPID[7615] ]]]
PID: 7615
[[[ This is 00-fork: PID[7615] PPID[7614] ]]]
PID: 7614
PID: 7614
>>>>> $
```

```
/*
 * Copyright (C) 2015-2020 Rahmat M. Samik-Ibrahim
 * http://rahmatm.samik-ibrahim.vlsm.org/
 * This program is free script/software.
 *
 * REV10 Tue Mar 24 16:38:29 WIB 2020
 * START Xxx Xxx XX XX:XX:XX XXX XXXX
 */

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

void main(void) {
    pid_t  pid1, pid2, pid3;

    pid1 = pid2 = pid3 = getpid();
    printf(" 2016   2015   Lainnya\n=====\\n");
    printf("[%5.5d] [%5.5d] [%5.5d]\\n", pid1, pid2, pid3);
    fork();
    pid1 = getpid();
    wait(NULL);
    pid2 = getpid();
    if(!fork()) {
        pid2 = getpid();
        fork();
    }
    pid3 = getpid();
    wait(NULL);
    printf("[%5.5d] [%5.5d] [%5.5d]\\n", pid1, pid2, pid3);
}
```

```
/*  
# INFO: UTS 2016-1 (midterm)  
*/
```

```
$ ./13-uas161
```

```
2016    2015    Lainnya
```

```
=====
```

```
[14492] [14492] [14492]
```

```
[14493] [14494] [14495]
```

```
[14493] [14494] [14494]
```

```
[14493] [14493] [14493]
```

```
[14492] [14496] [14497]
```

```
[14492] [14496] [14496]
```

```
[14492] [14492] [14492]
```

```

/* Copyright (C) 2016-2020 Rahmat M. Samik-Ibrahim
 * http://rahmatm.samik-ibrahim.vlsm.org/
 * This program is free script/software. This program is distributed in the
 * hope that it will be useful, but WITHOUT ANY WARRANTY; without even the
 * implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * REV08 Tue Mar 24 16:40:28 WIB 2020
 * START Sun Dec 04 00:00:00 WIB 2016
 *
 * wait()      = suspends until its child terminates.
 * fflush()    = flushes the user-space buffers.
 * getppid()   = get parent PID
 * ASSUME pid >= 1000 ES pid > ppid **
 */

```

```

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#define NN 2

```

```

void main(void) {
    int ii, rPID, rPPID, id1000=getpid();
    for (ii=1; ii<=NN; ii++) {
        fork();
        wait(NULL);
        rPID = getpid()-id1000+1000; /* "relative" */
        rPPID=getppid()-id1000+1000; /* "relative" */
        if (rPPID < 1000 || rPPID > rPID) rPPID=999;
        printf("Loop [%d] - rPID[%d] - rPPID[%4d]\n", ii, rPID, rPPID);
        fflush(NULL);
    }
}

```

```
/*  
# INFO: UTS 2016-2 (midterm)  
*/  
  
$ ./14-uas162  
Loop [1] - rPID[1001] - rPPID[1000]  
Loop [2] - rPID[1002] - rPPID[1001]  
Loop [2] - rPID[1001] - rPPID[1000]  
Loop [1] - rPID[1000] - rPPID[ 999]  
Loop [2] - rPID[1003] - rPPID[1000]  
Loop [2] - rPID[1000] - rPPID[ 999]
```

```
/* Copyright (C) 2005-2020 Rahmat M. Samik-Ibrahim  
 * http://rahmatm.samik-ibrahim.vlsm.org/  
 * This program is free script/software.  
 * REV00 Wed May 3 17:07:09 WIB 2017  
 * START 2005  
 * fflush(NULL): flushes all open output streams  
 * fork():      creates a new process by cloning  
 * getpid():     get PID (Process ID)  
 * wait(NULL):   wait until the child is terminated  
 */
```

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

```
void main(void) {  
    int firstPID = (int) getpid();  
    int RelPID;  
  
    fork();  
    wait(NULL);  
    fork();  
    wait(NULL);  
    fork();  
    wait(NULL);  
  
    RelPID=(int)getpid()-firstPID+1000;  
    printf("RelPID: %d\n", RelPID);  
    fflush(NULL);  
}
```

```
/*  
# INFO: UTS 2017-1 (midterm)  
*/
```

```
$ ./15-uas171
```

```
RelPID: 1003
```

```
RelPID: 1002
```

```
RelPID: 1004
```

```
RelPID: 1001
```

```
RelPID: 1006
```

```
RelPID: 1005
```

```
RelPID: 1007
```

```
RelPID: 1000
```

```
$
```



```

/*
 * (c) 2017-2020 Rahmat M. Samik-Ibrahim
 * http://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 * REV03 Tue Mar 24 16:42:16 WIB 2020
 * REV02 Mon Dec 11 17:46:01 WIB 2017
 * START Sun Dec 3 18:00:08 WIB 2017
 */

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

#define LOOP 3
#define OFFSET 1000

void main(void) {
    int basePID = getpid() - OFFSET;

    for (int ii=0; ii < LOOP; ii++) {
        if(!fork()) {
            printf("PID[%d]-PPID[%d]\n",
                getpid() - basePID,
                getppid() - basePID);
            fflush(NULL);
        }
        wait(NULL);
    }
}

```

```
/*  
# INFO: UTS 2017-2 (midterm)  
*/
```

```
$ ./16-uas172  
PID[1001]-PPID[1000]  
PID[1002]-PPID[1001]  
PID[1003]-PPID[1002]  
PID[1004]-PPID[1001]  
PID[1005]-PPID[1000]  
PID[1006]-PPID[1005]  
PID[1007]-PPID[1000]  
$
```

Week 06: Check List (Deadline: Monday, 16-Nov-2020).

- ☐ **Starting Point:** <https://os.vlsm.org/>
- ☐ Week 06: Assignment (more details in **os06.pdf**).
 - ① Read: (OSC10 chapter 03 + chapter 04)
 - ② TBA.
- ☐ The "Assignment Day" is every Thursday morning.
- ☐ This page is <https://os.vlsm.org/Slides/check06.pdf>.

The End

- ☐ This is the end of the presentation.
- ☒ This is the end of the presentation.
 - This is the end of the presentation.