

# 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!

REV218 05-Feb-2020

# Operating Systems 2020-1

A [08-10, Rm 3114, Mo/We] — B/M [10:10-12, Rm 3114, Mo/We] — C [13-15, Rm 3114, Mo/We]

D [10-12, Rm 2307(Mo), Rm 3113(We)] — E [08-10, Rm 2307(Mo), Rm 3113(We)]

Week	Schedule	Topic	OSC10
Week 00	27 Jan - 02 Feb 2020	Overview 1, Virtualization & Scripting	Ch. 1, 2, 18.
Week 01	03 Feb - 09 Feb 2020	Overview 2, Virtualization & Scripting	Ch. 1, 2, 18.
Week 02	10 Feb - 16 Feb 2020	Security, Protection, Privacy, & C-language	Ch. 16, 17
Week 03	17 Feb - 23 Feb 2020	File System & FUSE	Ch. 13, 14, 15
Week 04	24 Feb - 01 Mar 2020	Addressing, Shared Lib, & Pointer	Ch. 9
Week 05	02 Mar - 08 Mar 2020	Virtual Memory	Ch. 10
Reserved	09 Mar - 13 Mar 2020	Q & E	
MidTerm	14-21 Mar 2020 (TBA)	MidTerm (UTS)	Subject to change.
Week 06	23 Mar - 31 Mar 2020	Concurrency: Processes & Threads	Ch. 3, 4
Week 07	01 Apr - 07 Apr 2020	Synchronization & Deadlock	Ch. 6, 7, 8
Week 08	08 Apr - 14 Apr 2020	Scheduling + W06/W07	Ch. 5
Week 09a	15 Apr - 19 Apr 2020	Storage, Firmware, Bootldr, & Systemd	Ch. 11
Week 09b	20 Apr - 26 Apr 2020	OnLine & CoLearnIng	
Week 10	27 Apr - 28 Apr 2020	Storage, Firmware, Bootldr, & Systemd	Ch. 11
Week 10	29 Apr - 05 May 2020	I/O & Programming	Ch. 12
Reserved	06 May - 10 May 2020	Q & A	
Final	11-18 May 2020 (TBA)	Final (UAS)	This schedule is subject to change.
Extra	25 Jun 2020	Extra assignment confirmation	

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

- ❑ **Text Book** — Any recent/decent OS book. Eg. (**OSC10**) Silberschatz et. al.: **Operating System Concepts**, 10<sup>th</sup> Edition, 2018. See also <http://codex.cs.yale.edu/avi/os-book/OS10/>.
- ❑ **Resources**
  - ❑ **All In One** — [BADAK.cs.ui.ac.id:///extra/](http://BADAK.cs.ui.ac.id:///extra/) (**FASILKOM only!**).
  - ❑ **Download Slides and Demos from GitHub.com**  
<https://github.com/UI-FASILKOM-OS/SistemOperasi/>
  - ❑ **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).
- ❑ **Try Demos**
  - ❑ Your own Ubuntu system.
  - ❑ Ubuntu on VirtualBox, or VMWare, or ...
  - ❑ Windows Subsystem for Linux (**Windows 10 only!**).
  - ❑ SSH to [BADAK.cs.ui.ac.id](http://BADAK.cs.ui.ac.id) (**FASILKOM only!**).

# Agenda I

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

# Agenda II

- 17 07-fork
- 18 08-fork
- 19 09-fork
- 20 10-fork
- 21 11-fork
- 22 12-fork
- 23 14-fork
- 24 15-fork
- 25 The End

# Week 06 Concurrency: Topics<sup>1</sup>

- 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)

---

<sup>1</sup>Source: ACM IEEE CS Curricula 2013

## Week 06 Concurrency: Learning Outcomes (1)<sup>1</sup>

- 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]

---

<sup>1</sup>Source: ACM IEEE CS Curricula 2013

## Week 06 Concurrency: Learning Outcomes (2)<sup>1</sup>

- 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]

---

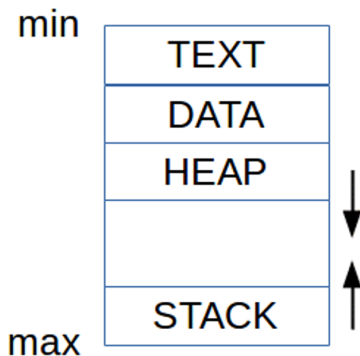
<sup>1</sup>Source: ACM IEEE CS Curricula 2013



# 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
P00=00-fork
P01=01-fork
...
P14=14-fork
P15=15-fork

EXECS= \
    $(P00) \
    $(P01) \
    ...
    $(P14) \
    $(P15) \

all: $(EXECS)

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

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

...

$(P14): $(P14).c
    $(CC) $(P14).c -o $(P14)

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

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

```
/*
 * (c) 2016-2017 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 * REV04 Mon Oct 30 10:28:12 WIB 2017
 * 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-fork: PID[%d] PPID[%d] ]]]\n",
           getpid(), getppid());
}

>>>> $ 00-fork

[[[ This is 00-fork: 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-sleeping

```
#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
```

# 05-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-fork: PID[6008] PPID[6007] ]]]
Parent:          PID[6007] PPID[1350] <<< <<< <<<
```

```
no execlp =====
Start:          PID[6040] PPID[1350]
Child:          PID[6041] PPID[6040] <<< <<< <<<
Parent:          PID[6040] PPID[1350] <<< <<< <<<
```

# 06-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==6072 ==== \n
VAL1= 0 VAL2= 0 VAL3= 0
VAL1= 0 VAL2= 0 VAL3=6075
VAL1= 0 VAL2=6074 VAL3= 0
VAL1= 0 VAL2=6074 VAL3=6076
VAL1=6073 VAL2= 0 VAL3= 0
VAL1=6073 VAL2= 0 VAL3=6078
VAL1=6073 VAL2=6077 VAL3= 0
VAL1=6073 VAL2=6077 VAL3=6079
```

# 07-fork

```
>>>> $ cat 07-fork.c
/*
 * (c) 2005-2017 Rahmat M. Samik-Ibrahim
 * https://rahmatm.samik-ibrahim.vlsm.org/
 * This is free software.
 * REV05 Mon Oct 30 10:57:02 WIB 2017
 * REV02 Mon Oct 24 10:43:00 WIB 2016
 * REV01 Sun Feb 27 08:31:46 WIB 2011
 * START 2005
 */

#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define DISPLAY1 "START * PARENT *** ** PID (%4d) ** *****\n"
#define DISPLAY2 "RANDOM: val1(%4d) -- val2(%4d) -- val3(%4d)\n"
/***** main ** */
void main(void) {
    pid_t val1, val2, val3;
    printf(DISPLAY1, getpid());
    val1 = fork();
    val2 = fork();
    val3 = fork();
    printf(DISPLAY2, val1, val2, val3);
    wait(NULL);
    wait(NULL);
    wait(NULL);
    /* ***** START BLOCK ***
     ***** END * BLOCK *** */
}
```



## 07-fork (2)

```
>>>> $ 07-fork
START * PARENT *** ** PID (6160) ** *****
RANDOM: val1(6161) -- val2(6162) -- val3(6163)
RANDOM: val1(6161) -- val2(6162) -- val3(  0)
RANDOM: val1(6161) -- val2(  0) -- val3(6165)
RANDOM: val1(6161) -- val2(  0) -- val3(  0)
RANDOM: val1(  0) -- val2(6164) -- val3(6166)
RANDOM: val1(  0) -- val2(6164) -- val3(  0)
RANDOM: val1(  0) -- val2(  0) -- val3(6167)
RANDOM: val1(  0) -- val2(  0) -- val3(  0)
>>>> $ 07-fork
START * PARENT *** ** PID (6168) ** *****
RANDOM: val1(6169) -- val2(6170) -- val3(6172)
RANDOM: val1(6169) -- val2(  0) -- val3(6173)
RANDOM: val1(6169) -- val2(6170) -- val3(  0)
RANDOM: val1(  0) -- val2(6171) -- val3(6174)
RANDOM: val1(6169) -- val2(  0) -- val3(  0)
RANDOM: val1(  0) -- val2(  0) -- val3(6175)
RANDOM: val1(  0) -- val2(  0) -- val3(  0)
RANDOM: val1(  0) -- val2(6171) -- val3(  0)
>>>> $ 07-fork
START * PARENT *** ** PID (6176) ** *****
RANDOM: val1(6177) -- val2(6178) -- val3(6181)
RANDOM: val1(  0) -- val2(6179) -- val3(6180)
RANDOM: val1(  0) -- val2(6179) -- val3(  0)
RANDOM: val1(  0) -- val2(  0) -- val3(6182)
RANDOM: val1(6177) -- val2(  0) -- val3(6183)
RANDOM: val1(6177) -- val2(  0) -- val3(  0)
RANDOM: val1(6177) -- val2(6178) -- val3(  0)
RANDOM: val1(  0) -- val2(  0) -- val3(  0)
>>>> $
```

# 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
>>>>> $
```

# 14-fork

```
#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);
}

=====
RelPID: 1003
RelPID: 1002
RelPID: 1004
RelPID: 1001
RelPID: 1006
RelPID: 1005
RelPID: 1007
RelPID: 1000
>>>> $
```



# 15-fork

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

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

=====
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]
>>>> $
```

# The End

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