

Fork, Wait

Praktikum Sistem Operasi

Ilmu Komputer IPB

2017

fork()

Fungsi fork()

```
pid_t fork(void);
```

- ▶ Menduplikasi proses¹
- ▶ *Return value:*
 - ▶ jika *parent*: PID *child*
 - ▶ jika *child*: 0
 - ▶ jika *error*: -1

¹lihat 'man 2 fork'

Contoh

```
// fork.c
#include <stdio.h>
#include <unistd.h>

int main()
{
    fork();
    puts("hello");
    return 0;
}
```

Latihan

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    fork();
    fork();
    puts("X");
    return 0;
}
```

Berapa kali X dicetak?

Parent atau *Child*?

Return value `fork()` digunakan untuk membedakan antara proses *parent* dengan *child*.

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

int main()
{
    pid_t pid;                // process id

    pid = fork();
    if (pid == 0)
        puts("child");
    else
        puts("parent");

    return 0;
}
```

Konkurensi

- ▶ Proses *parent* dan *child* berjalan secara konkuren.
 - ▶ meskipun pada kode program terlihat sekuensial
- ▶ Proses *child* memiliki semua salinan variabel *parent*-nya.


```
int main() {
    pid_t pid; char *msg; int n;

    pid = fork();
    if (pid == 0) {
        msg = "child";
        n = 4;
    } else {
        msg = "parent";
        n = 8;
    }

    while (n-- > 0) { puts(msg); sleep(1); }
    return 0;
}
```

Latihan

Dengan menggunakan *looping* dan proses *parent* tidak mencetak apapun:

- ▶ Buatlah 4 proses *child* untuk mencetak “Hello”!
- ▶ Buatlah 5 proses *child* untuk mencetak “Hello”!
- ▶ Buatlah 9 proses *child* untuk mencetak “Hello”!

wait()

Fungsi wait()

```
pid_t wait(int *status);
```

- ▶ Proses *parent* menunggu hingga salah satu proses *child* selesai².
 - ▶ jika *child* sudah selesai, semua *resource*-nya akan dilepaskan
 - ▶ lalu proses *parent* melanjutkan eksekusi proses
- ▶ *Return value*: PID *child*.
- ▶ Argumen *status*: menyimpan *exit status* proses *child*
 - ▶ isi dengan NULL jika tidak dipakai

²lihat 'man 2 wait'

Contoh

```
int main() {  
    pid_t pid; char *msg; int n;  
  
    pid = fork();  
    if (pid == 0) { msg = "child"; n = 8; }  
        else { msg = "parent"; n = 4; }  
  
    while (n--) { puts(msg); sleep(1); }  
  
    if (pid > 0)    // parent menunggu child selesai  
        wait(NULL);  
  
    return 0;  
}
```

Mendapatkan Status Proses *Child*

Gunakan parameter status pada fungsi wait() untuk menyimpan *exit status* proses *child* yang telah selesai.

```
if (pid > 0) {  
    int status;  
    wait(&status);  
    printf("child status: %d\n", WEXITSTATUS(status));  
}
```

Zombie

Proses Zombie

- ▶ Proses *child* sudah selesai, tetapi masih ada di memori.
- ▶ Sebab: proses *child* sudah selesai, tetapi *parent* masih berjalan dan tidak memanggil fungsi `wait()`.

Contoh

```
// zombie.c
int main()
{
    pid_t pid; char *msg; int n;

    pid = fork();
    if (pid == 0) { msg = "child"; n = 3;}
        else { msg = "parent"; n = 20;}

    while (n-->0) { puts(msg); sleep(1); }

    return 0;
}
```

Mengamati Zombie

- ▶ Jalankan program tersebut pada terminal.
- ▶ Buka terminal baru, jalankan `ps f` untuk melihat daftar proses.
 - ▶ lakukan dengan cepat, zombie hanya muncul selama 17 detik
- ▶ Ciri zombie: status Z dan nama proses <defunct>.

PID	TTY	STAT	TIME	COMMAND
12027	pts/1	Ss	0:00	bash
12028	pts/1	S+	0:00	_ ./zombie
12029	pts/1	Z+	0:00	_ [zombie] <defunct>
12031	pts/2	Ss	0:00	bash
12032	pts/2	R+	0:00	_ ps f

Latihan

// Berapa kali X dicetak, gambarkan pohon prosesnya!

```
int main()
{
    pid_t pid = fork();
    if (pid != 0)
        fork();
    fork();
    puts("X");
    return 0;
}
```

Latihan Buku

- ▶ Kerjakan latihan (buku hlm 149–153) nomor:
 - ▶ 3.1
 - ▶ 3.2
 - ▶ 3.12
 - ▶ 3.13
 - ▶ 3.14
 - ▶ 3.17
- ▶ Kerjakan tanpa bantuan komputer terlebih dahulu, lalu cek jawaban anda dengan menjalankan kode program yang diberikan.

Latihan Tambahan

- ▶ Kerjakan soal nomor:
 - ▶ 3.21
 - ▶ 3.22 (baca sendiri ttg *shared-memory* antarproses)