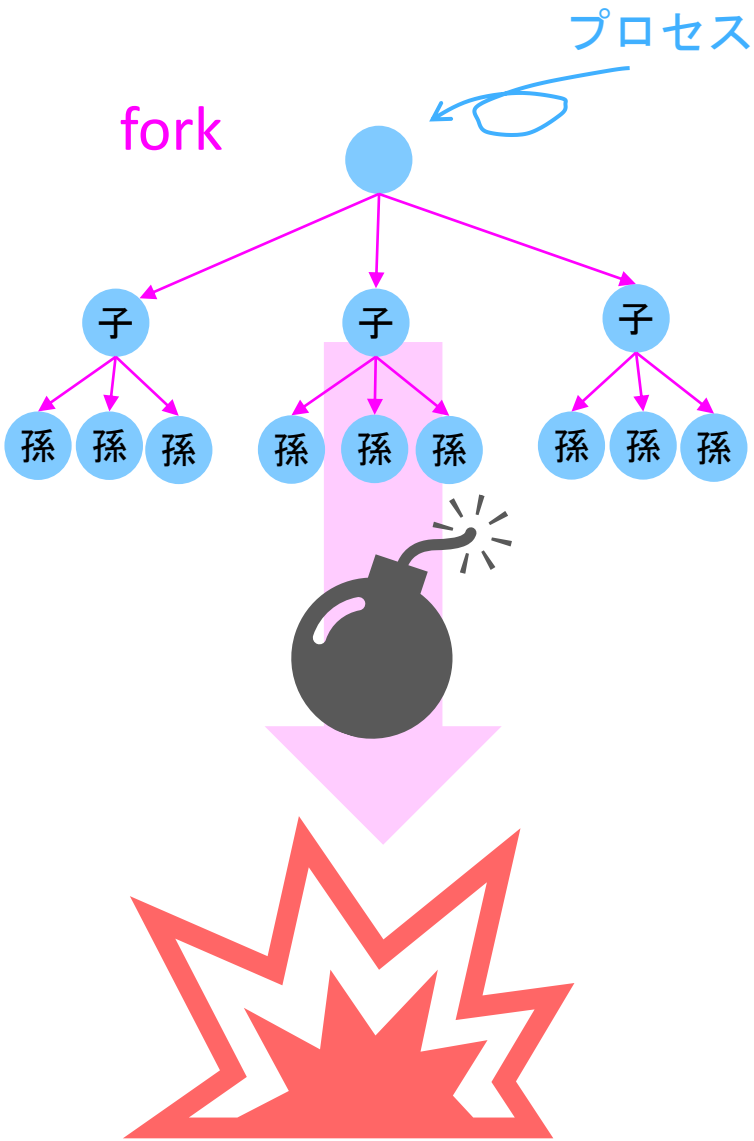
Fork Bomb (Fork爆弾)とは



●DoS (過剰に負荷をかける)攻撃
- 高速に多数のプロセスを生成する
- プロセステーブルを埋めつくす
- 新たなプロセスを生成できなくする
- 対処するリソースを取れなくする
- SSH接続や操作ができなくなる

● 原因
- 悪意ある攻撃
- プログラミングのミス
  - ginでは注意すること

● 対策

ユーザが生成できるプロセス数の制限
- 通常の挙動で超えることも
- 複数人の協働やスペック次第で突破

```
$ ulimit -u
3870
```

CentOS 7 の場合

Fork Bombの実行時

```
top - 23:11:40 up  3:43,  2 users,  load average: 1262.67, 292.69, 96.62
Tasks: 4172 total,  95 running, 4077 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.6 us, 10.6 sy,  0.0 ni,  9.5 id, 77.8 wa,  0.0 hi,  1.4 si,  0.0 st
KiB Mem :  1014296 total,   464360 free,   378132 used,   171804 buff/cache
KiB Swap:   978940 total,   978940 free,        0 used.   490392 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 5791 itakeha+  20   0    4216     20      0 D   3.7  0.0   0:00.11 a.out
 4735 itakeha+  20   0    4216     20      0 D   2.7  0.0   0:00.08 a.out
 4863 itakeha+  20   0    4216     20      0 D   2.3  0.0   0:00.07 a.out
 4999 itakeha+  20   0    4216     20      0 D   2.3  0.0   0:00.07 a.out
 2240 itakeha+  20   0    4216     32      4 D   2.0  0.0   0:00.06 a.out
 4837 itakeha+  20   0    4216     20      0 D   2.0  0.0   0:00.06 a.out
 2627 itakeha+  20   0    4216     44      0 D   1.7  0.0   0:00.05 a.out
 4305 itakeha+  20   0    4216     28      0 D   1.7  0.0   0:00.05 a.out
 4660 itakeha+  20   0    4216     28      0 D   1.7  0.0   0:00.05 a.out
 4784 itakeha+  20   0    4216     28      0 D   1.7  0.0   0:00.05 a.out
…
```

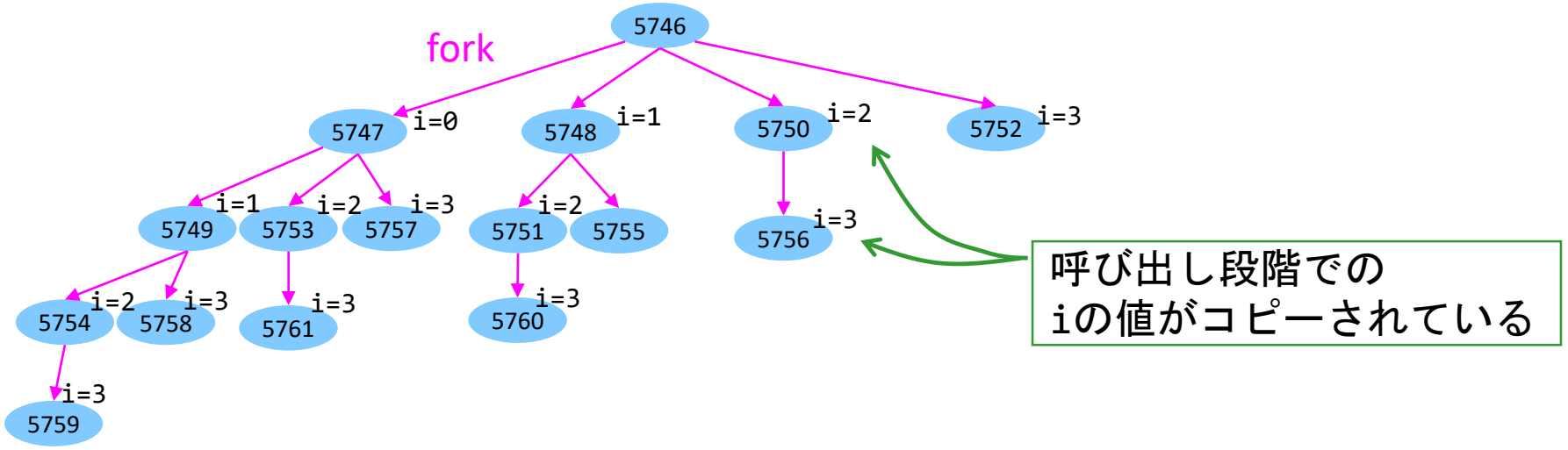
プロセスが大量に生成されている

```
$ top
-bash: fork: retry: No child processes
-bash: fork: retry: No child processes
-bash: fork: retry: No child processes
-bash: fork: retry: No child processes
-bash: fork: Resource temporarily unavailable
```

新たにプロセスが生成できない

Fork Bomb前からSSHでアクセスしていた

プロセスを増やして遊ぼう！




```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#define CNUM 4 // Child Process Nums

int main(void){
    int i,st;
    printf("host pid:%2d¥n",getpid());
    for(i=0;i<CNUM;i++){
        int cpid=fork();
        if(cpid!=0){ // Parent Process
            printf("count:%2d, parent:%5d,child:%5d¥n",i,getpid(),cpid);
        }
    }
    wait(&st);
    return 0;
}
```

while(1){

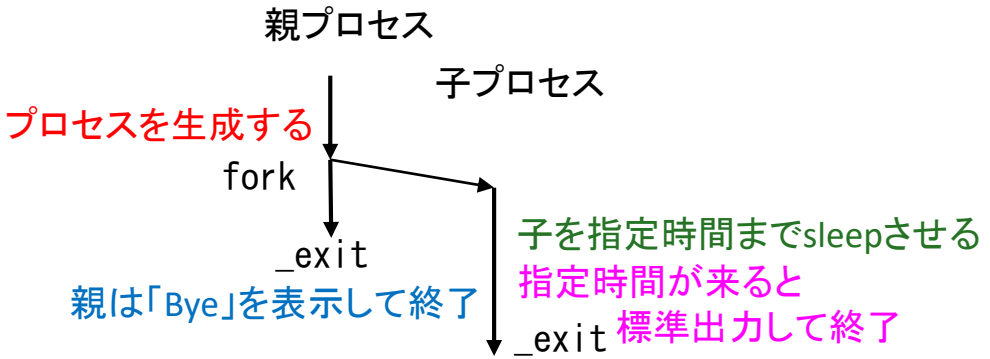
```
host pid:5746
count: 0, parent: 5746,child: 5747
count: 1, parent: 5746,child: 5748
count: 2, parent: 5746,child: 5750
count: 3, parent: 5746,child: 5752
count: 1, parent: 5747,child: 5749
count: 2, parent: 5748,child: 5751
count: 2, parent: 5747,child: 5753
count: 3, parent: 5748,child: 5755
count: 2, parent: 5749,child: 5754
count: 3, parent: 5750,child: 5756
count: 3, parent: 5747,child: 5757
count: 3, parent: 5749,child: 5758
count: 3, parent: 5754,child: 5759
count: 3, parent: 5751,child: 5760
count: 3, parent: 5753,child: 5761
```

forkを用いたプログラム

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

int main(int argc,char **argv){
    int min=1;    // 指定時間(分)
    int sec;      // 指定時間(秒)
    pid_t pid;    // プロセスID

    sec=min*60;

    pid=fork();
    if(pid==0){
        sleep(sec);
        printf("Progress %2d min!¥n",min);
        exit(0);
    }
    puts("Bye");

    return 0;
}
```

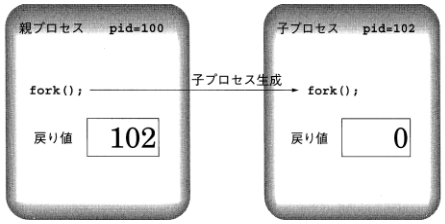
指定時間になったら「Progress ~ min!」というメッセージを表示する

- プロセスを生成する
- 子を指定時間までsleepさせる
- 指定時間が来ると標準出力して終了
- 親は「Bye」を表示して終了




```c
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

- 失敗すれば　-1　が返る
- 成功すれば,
  - 親プロセスには子プロセスのPIDが返る
  - 子プロセスには0が返る



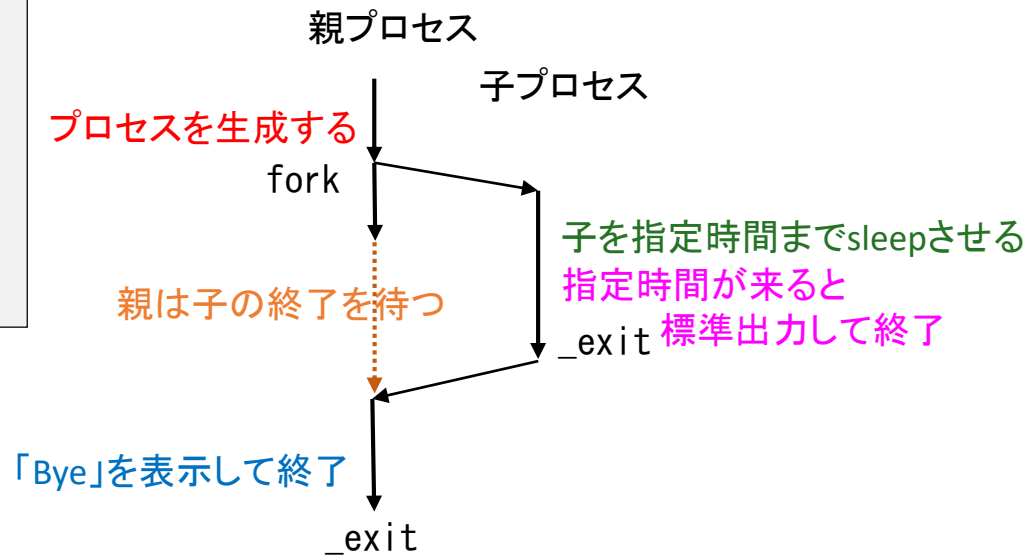
埴井 正雄. “Linuxシステムプログラミング”, 株式会社オーム社, pp.24--25, 平成16年7月20日 第1版
Linuxシステムプログラミング | Ohmsha, https://www.ohmsha.co.jp/book/9784274079726/, 2021/05/24

forkとwaitを用いたプログラム

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

int main(int argc,char **argv){
    int min=1;
    int sec;
    pid_t pid;
    int st;

    sec=min*60;

    pid=fork();
    if(pid==0){
        sleep(sec);
        printf("Progress %2d min!¥n",min);
        exit(0);
    }
    wait(&st);
    puts("Bye");

    return 0;
}
```

指定時間になったら「Progress ~ min!」というメッセージを表示する

- プロセスを生成する
- 子を指定時間までsleepさせる
- 指定時間が来ると標準出力
- 親は子の終了を待つ
- 「Bye」を表示して終了



埴井 正雄. “Linuxシステムプログラミング”, 株式会社オーム社, pp.24--25, 平成16年7月20日 第1版
Linuxシステムプログラミング | Ohmsha, https://www.ohmsha.co.jp/book/9784274079726/, 2021/05/24

forkの特徴

● 呼び出したプロセス自身のコピーを作成する
- 親子を分けて異なる挙動を記述する
- forkの返り値を利用する

● オリジナルとコピーでそれぞれ異なる仮想アドレス空間を持つ
- 親(オリジナル)子(コピー)で相互にデータをやり取りできない
  - 子で値を変更しても書き込むことができない
- 共有メモリ(Shared Memory)を使う
  - 大学院講義「コンカレントプログラミング」

```
#define VAR_DUMP(X) { printf("data(%2d) is %p¥n",X,&X);}
int main(void)
{
    int data=0,st;
    pid_t pid=fork();

    if(pid==0){
        data=10;
        VAR_DUMP(data);
        exit(0);
    }

    wait(&st);
    VAR_DUMP(data);
    return 0;
}
```

```
$ ./a.out
data(10) is 0x7ffc65fb3cb8
data( 0) is 0x7ffc65fb3cb8
```

同じアドレスにあるハズの変数dataを子プロセスから変更できない

バッファオーバーフローを悪用する

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

#define VAR_DUMP(X) ¥
    { printf(“%7s,(%c) is %p¥n”,#X,X,&X);}

int main(int argc,char *argv[])
{
    char cmd[2],user[8];
    strncpy(cmd,"ls”,2);

    printf("Input Your Name");
    gets(user);

    printf("Input Commnad is %s by %s¥n",cmd,user);
    puts("=== Result ===");
    system(cmd);
    puts("=== ===");

    return 0;
}
```

中身とアドレスを検証

cmdの中身も書き換わる

- gets関数とsystem関数を悪用
  - 標準入力からユーザ名を入れる
  - ``ls’’をsystem関数で実行する
  - system関数の結果とユーザ名を表示する

```
$ ./a.out
Input Your Name: s20g470
Input Commnad is ls by s20g470
=== Result ===
a.out  overflow.c
=== ===
```

想定している実行結果

```
$ ./a.out
Input Your Name: abcdef1234567890pwd
Input Commnad is pwd by abcdef1234567890pwd
=== Result ===
/home/itakehara/work
=== ===
```

悪用した実行結果

メモリを壊してみましょう | 学校では教えてくれないこと | [技術コラム集]組込みの門 | ユークエスト株式会社, https://www.uquest.co.jp/embedded/learning/lecture08.html, 2021/05/25