Topic 2

Parent/child Process and Related C Functions

P1 – Pre Group 10

Parent/child Process

- process: program under execution
- parent process: process make fork() call
- child process: process created by the fork() call.
- Once a new child process is created, both processes will execute the next instruction following the fork() system call.
- A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

fork()

- pid_t fork(void);
- Below are different values returned by fork().
  - Negative Value: creation of a child process was unsuccessful.
  - Zero: Returned to the newly created child process.
  - Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

fork()

- if pid < 0
  - fail to fork, exit
- if pid == 0:
  - child process
- if pid >0:
  - parent process

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

int main ()
{
    pid_t fpid;
    fpid=fork();
```

```c
    if (fpid < 0)
        printf("error in fork!");
    else if (fpid == 0) {
        printf("i am the child process\n");
    }
    else {
        printf("i am the parent process\n");
    }

    return 0;
}
```

Parent process pid > 0

```c
    if (fpid < 0)
        printf("error in fork!");
    else if (fpid == 0) {
        printf("i am the child process\n");
    }
    else {
        printf("i am the parent process\n");
    }

    return 0;
}
```

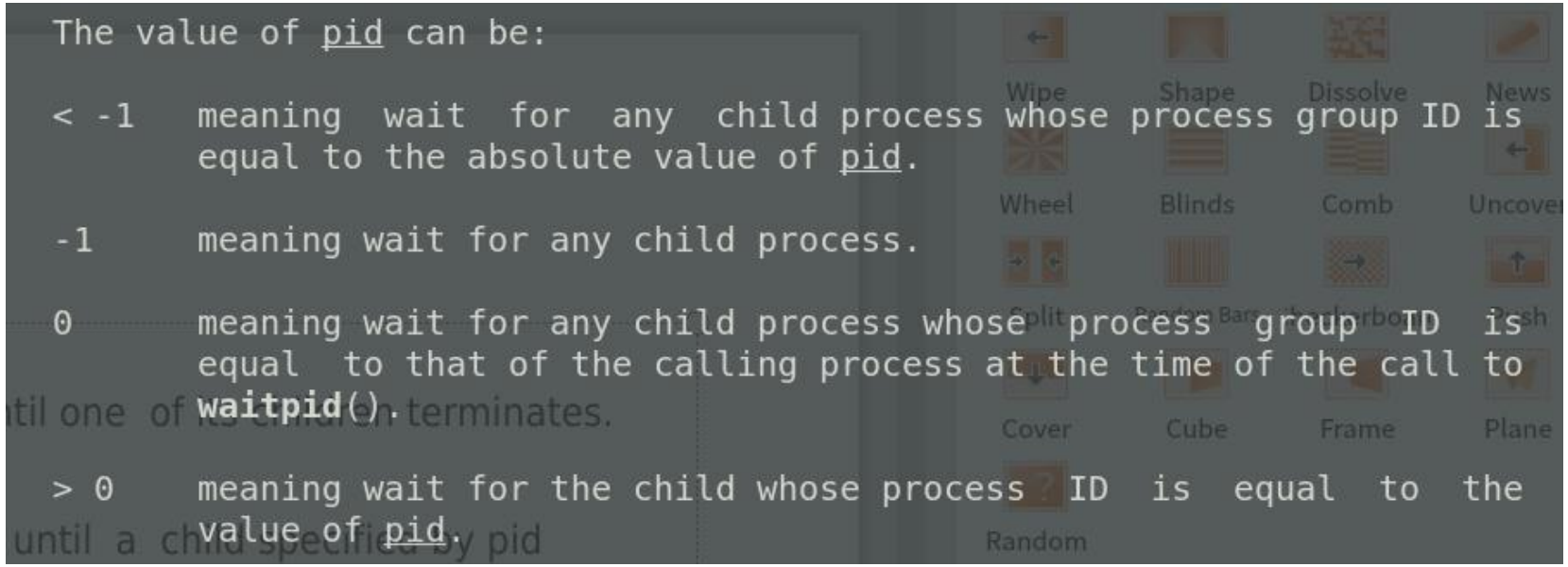
Children process pid < 0

wait(),waitpid()

- used to wait for state changes in a child process, and obtain information about the child process whose state has changed.
- A state change is considered to be: terminated; stopped by a signal; or was resumed by a signal.
- In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then the terminated child remains in a "zombie" state .
- If a child has already changed state, then these calls return immediately. Otherwise, they block until either a child changes state or a signal handler interrupts the call

wait(),waitpid()

- pid_t wait(int *wstatus);
- wait() suspends execution of the calling thread until one of its children terminates.
- pid_t waitpid(pid_t pid, int *wstatus, int options);
- waitpid() suspends execution of the calling thread until a child specified by pid argument has changed state. By default, waits only for terminated children, but can be modifiable via the options argument
- waitpid(-1, &wstatus, 0); == wait(&wstatus);



wait(),waitpid()

- you can use macros to inspect the integer wstate to get more information
- WIFEXITED(wstatus)
- returns true if the child terminated normally, that is, by call-
- ing exit(3) or _exit(2), or by returning from main().
- WIFSIGNALED(wstatus)
- returns true if the child process was terminated by a signal.

Thanks!