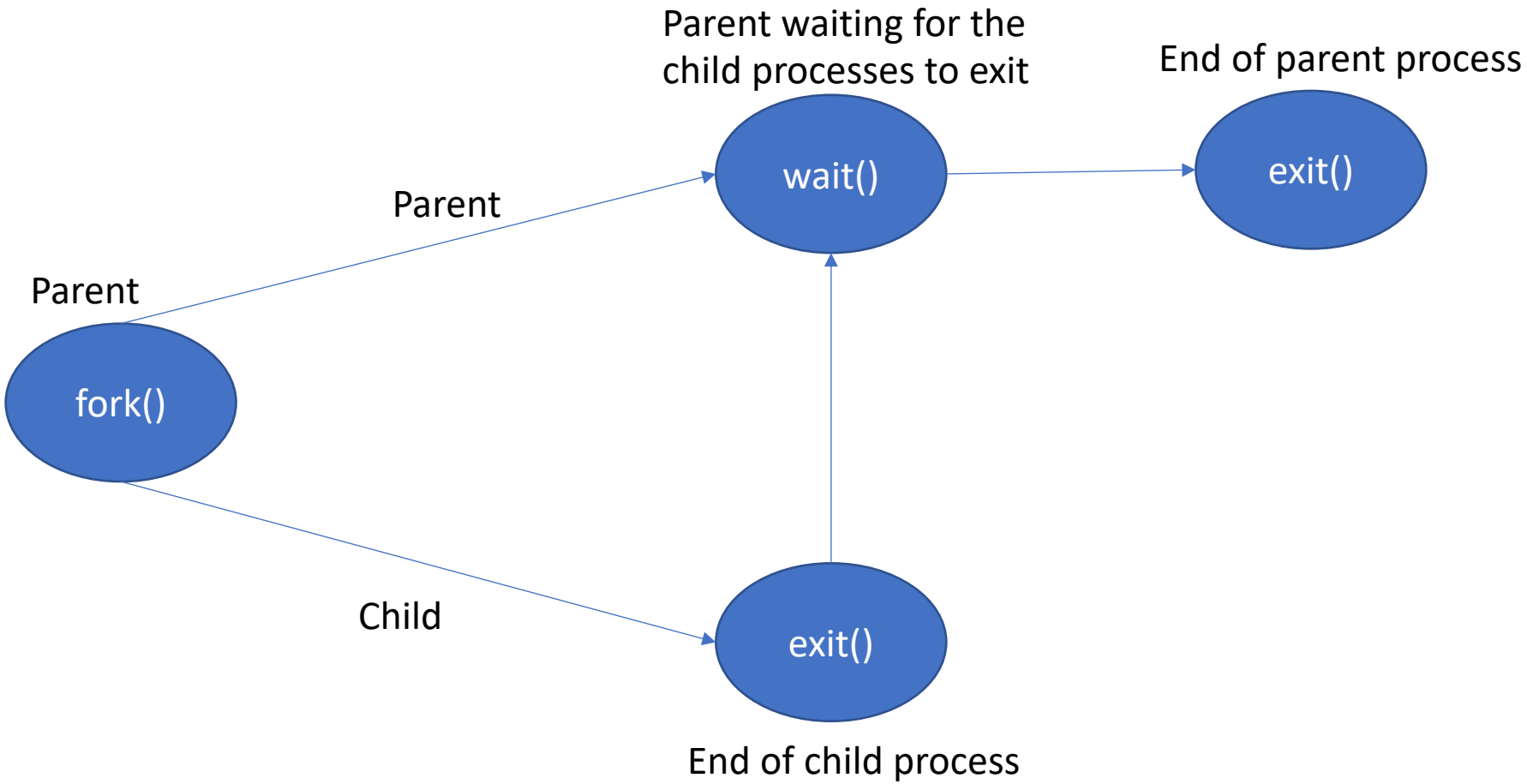
Read chapter1 from txv6 book: “Operating system organization”

System call	Description
fork()	Create a process
exit()	Terminate the current process
wait()	Wait for a child process to exit
kill(pid)	Terminate process pid
getpid()	Return the current process’s pid
sleep(n)	Sleep for n clock ticks
exec(filename, *argv)	Load a file and execute it
sbrk(n)	Grow process’s memory by n bytes
open(filename, flags)	Open a file; the flags indicate read/write
read(fd, buf, n)	Read n bytes from an open file into buf
write(fd, buf, n)	Write n bytes to an open file
close(fd)	Release open file fd
dup(fd)	Duplicate fd
pipe(p)	Create a pipe and return fd’s in p
chdir(dirname)	Change the current directory
mkdir(dirname)	Create a new directory
mknod(name, major, minor)	Create a device file
fstat(fd)	Return info about an open file
link(f1, f2)	Create another name (f2) for the file f1
unlink(filename)	Remove a file


Parent waiting for the
child processes to exit
End of parent process
wait()
exit()
Parent
Parent
fork()
Child
exit()
End of child process

```c
#include "types.h"
#include "user.h"

int main(int argc, char *argv[])
{
  int pid, ret_pid, exit_status;

  pid = fork();
//-----------------------------------------------------------------
    if (pid == 0) { // only the child executed this code
        printf(1, "\nThis is child with PID# %d and I will exit with status %d\n", getpid(), 0);
        exit(-1);   Should have been exit(0)
    }
//-----------------------------------------------------------------
    else if (pid > 0) { // only the parent executes this code
        ret_pid = wait(&exit_status);
         printf(1, "\n This is the parent: child with PID# %d has exited with status %d\n",
ret_pid, exit_status);   exit_status now contains 0 which was exit status returned by child.
    }
    else
        exit(-1);
// now it is time for the parent processes to exit.
exit(0);
return 0;
 }
```

Parent code

```
#include "types.h"
#include "user.h"

int main(int argc, char *argv[])
{
  int pid, ret_pid, exit_status;

  pid = fork();
if (pid > 0) { // only the parent executes this code
            ret_pid = wait(&exit_status);
             printf(1, "\n This is the parent: child with PID# %d has
exited with status %d\n", ret_pid, exit_status);
    }
    else
            exit(-1);
// now it is time for the parent processes to exit.
exit(0);
return 0;
 }
```

Child code

```
#include "types.h"
#include "user.h"

int main(int argc, char *argv[])
{
  int pid, ret_pid, exit_status;

  pid = fork();
//-----------------------------------------------------------------------
    if (pid == 0) { // only the child executed this code
            printf(1, "\nThis is child with PID# %d and I will exit with
status %d\n", getpid(), 0);
            exit(-1);
    }
```