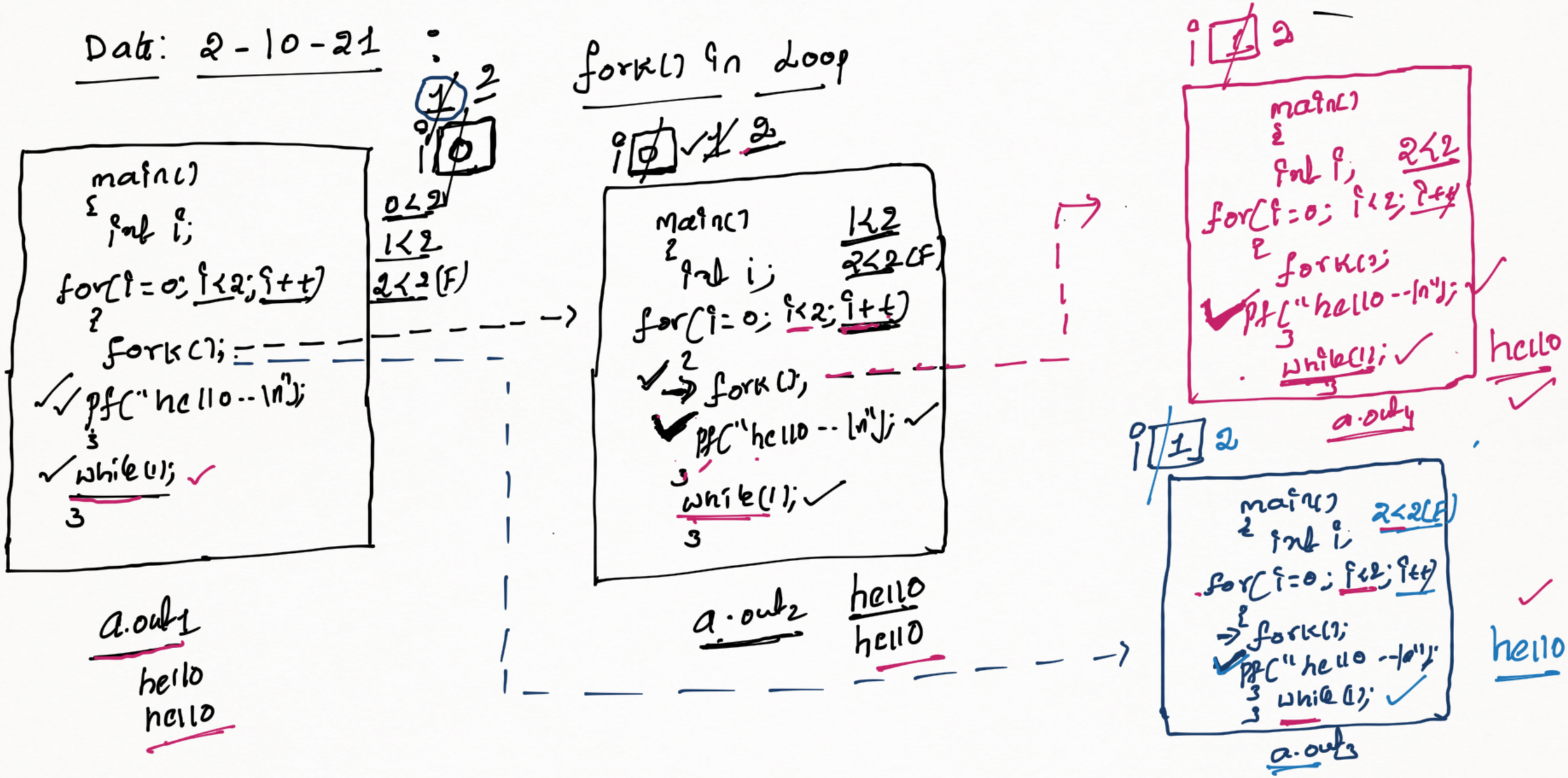
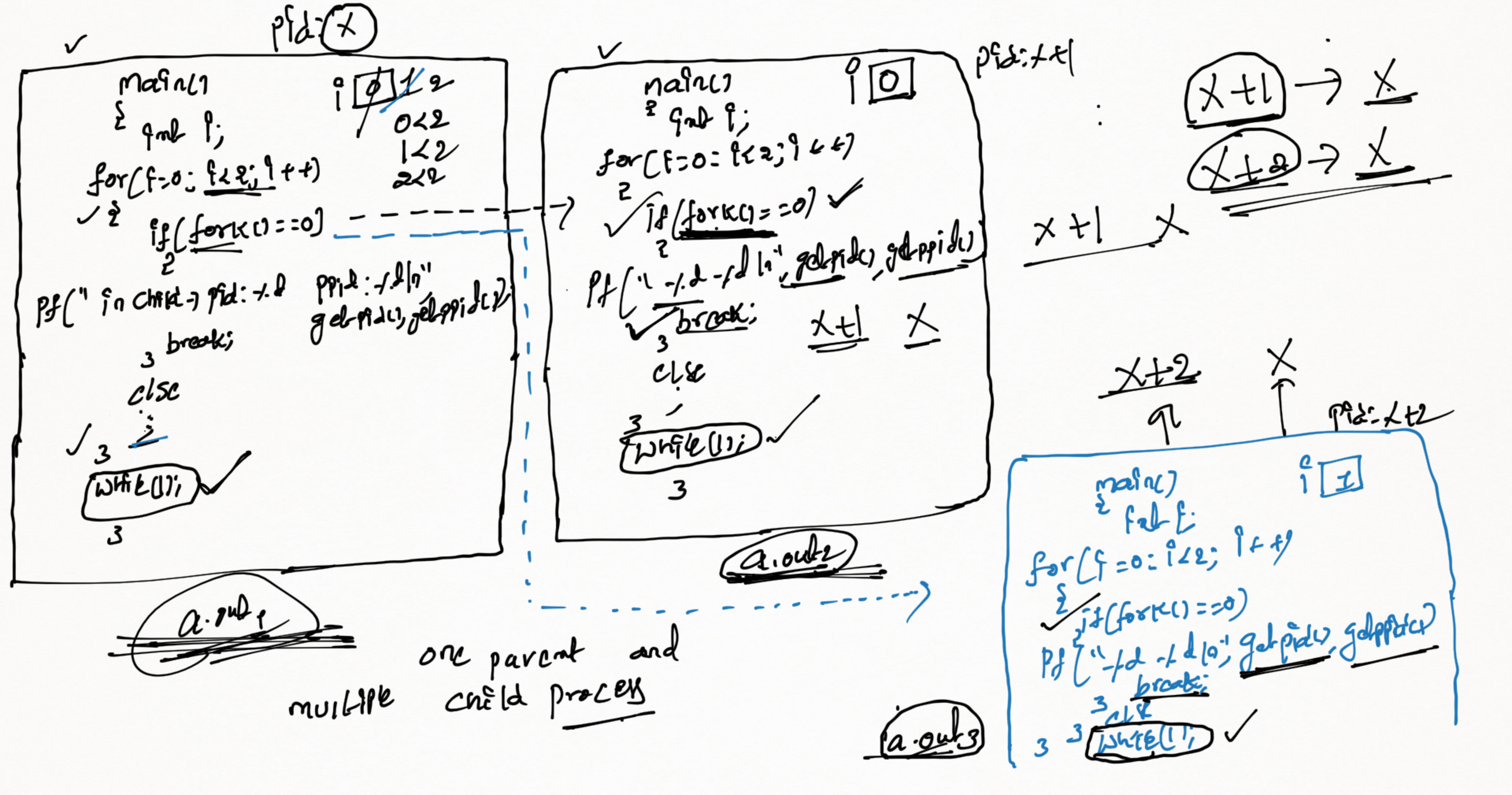
Date: 2-10-21

fork() in loop




pid: X
main()
{ int i;
for(i=0; i<2; i++)
if(fork()==0)
printf("I am child -> pid: %d ppid: %d\n", getpid(), getppid());
break;
else
while(1);
i 0 1 2
0<2
1<2
2<2
a.out1
pid: X+1
main()
{ int i;
for(i=0; i<2; i++)
if(fork()==0)
printf("I am child -> pid: %d ppid: %d\n", getpid(), getppid());
break;
else
while(1);
i 0
X+1 X
a.out2
pid: X+2
main()
{ int i;
for(i=0; i<2; i++)
if(fork()==0)
printf("I am child -> pid: %d ppid: %d\n", getpid(), getppid());
break;
else
while(1);
i 1
X+2 X
a.out3
X+1 → X
X+2 → X
multiple child process

When fork() fails

$ vi proc/sys/kernel/pid_max

32768
↳ 0 To 32767

→ For every operating system there is a limit on how many processes it can handle and also there is a limit for each user how many process possible to create once the number of processes reaching to that limit then new process creation not allowed in this case fork() fails and returns -1

system Vs fork

→ system function creates a new process by calling sh shell internally.

→ system function execution follows sequential

→ since for every process execution sh shell is executed so no. of processes created are more

→ fork creates a new process, (called child process) and follows parallel execution by taking context switch.

→ Using fork() no. of process are less compared to system

→ Fork supports COW mechanism (copy-on-write)