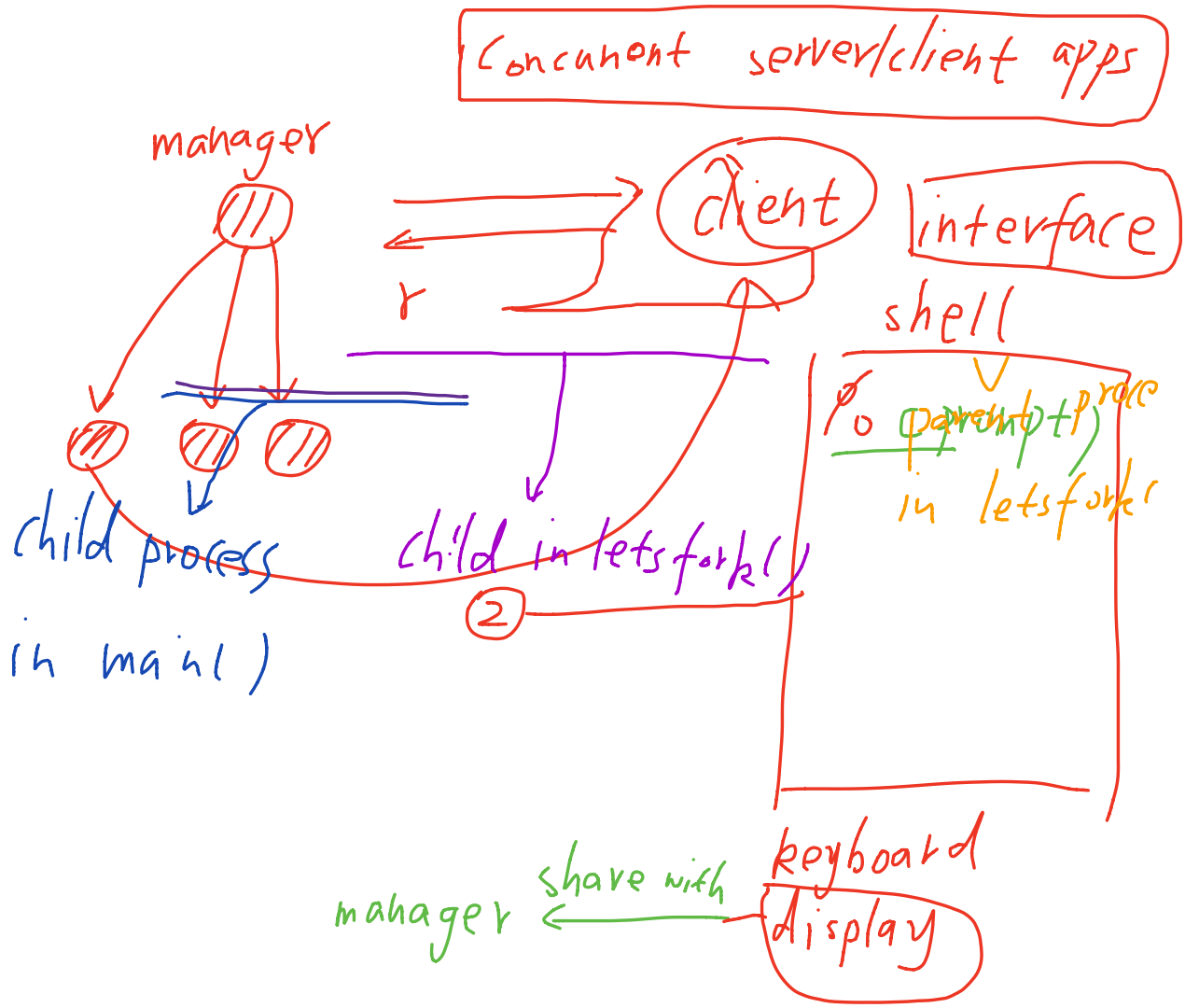
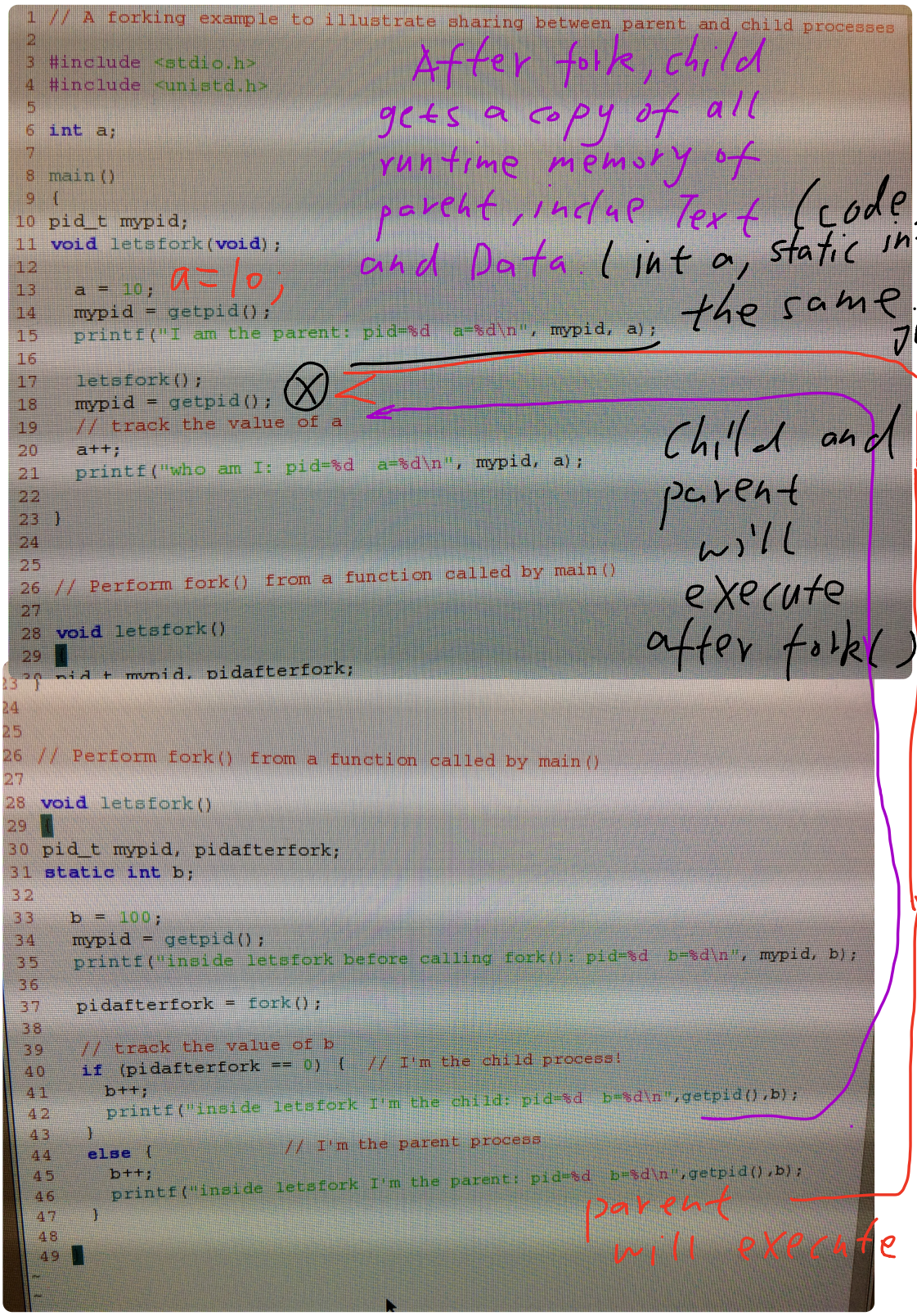

Concurrent server/client apps
manager
client
interface
r
shell
child process
in main()
child in letsfork()
2
keyboard
display
manager
share with



```c
// A forking example to illustrate sharing between parent and child processes

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

int a;

main()
{
pid_t mypid;
void letsfork(void);

   a = 10;
   mypid = getpid();
   printf("I am the parent: pid=%d  a=%d\n", mypid, a);

   letsfork();
   mypid = getpid();
   // track the value of a
   a++;
   printf("who am I: pid=%d  a=%d\n", mypid, a);

}


// Perform fork() from a function called by main()

void letsfork()
{
pid_t mypid, pidafterfork;
static int b;

   b = 100;
   mypid = getpid();
   printf("inside letsfork before calling fork(): pid=%d  b=%d\n", mypid, b);

   pidafterfork = fork();

   // track the value of b
   if (pidafterfork == 0) {  // I'm the child process!
     b++;
     printf("inside letsfork I'm the child: pid=%d  b=%d\n",getpid(),b);
   }
   else {                    // I'm the parent process
     b++;
     printf("inside letsfork I'm the parent: pid=%d  b=%d\n",getpid(),b);
   }

}
```

After fork, child gets a copy of all runtime memory of parent, include Text (code) and Data. (int a, static int b)

a=10;

the same function

Child and parent will execute after fork()

parent will execute

parent process in main()

parent process in main()

```
gcc main1.c
./a.out
I am the parent: pid=28099  a=10
inside letsfork before calling fork(): pid=28099  b=100
inside letsfork I'm the parent: pid=28099  b=100
who am I: pid=28099  a=11
inside letsfork I'm the child: pid=28100  b=101
who am I: pid=28100  a=11
```

parent process in letsfork()

child process in main()

child in letsfork()

After fork, child will have a copy of parent memory, including

Data = (int a, static int b)

uninitialized global ; initialized static : Data section)

And Run the same code as parent.

→ In Parent proces (pid afterfor != 0)

In parent process letsfork() | b++ ; // parent process static int b=100

printf (getpid() , b); → (P-b)

P_letsfork()

parent pid=28099 , P-b=101

then parent process return from function
letsfork(), execute next line of
In main() {
In parent main()

letsfork();
mypid = getpid(); → pid=2809
p_mypid ←
att; (parent global int a)
p_a = 11
printf(mypid, a)
p_mypid
p_a
in parent process stack
in parent process BSS area;

The parent process exit from
parent process main(); P_main()

In child process, first in child process's
letsfork() → c_letsfork
(pidafterfork == 0) → child process pidafterfork
c_pidafterfork

bff; → child process static int b

printf(getpid, b);

c_b

c_b=100;
c_bff;
c_b=101;

child process pid=28100

// will print 28100, 101

The child process exit from his letsfork() function, and return to the next line after

of his own main()

c_main()

in main(){

letsfork();

child process main code

mypid = getpid();

child process mypid, in child process stack;

aff;

printf(mypid, a)

c_mypid =28100

}

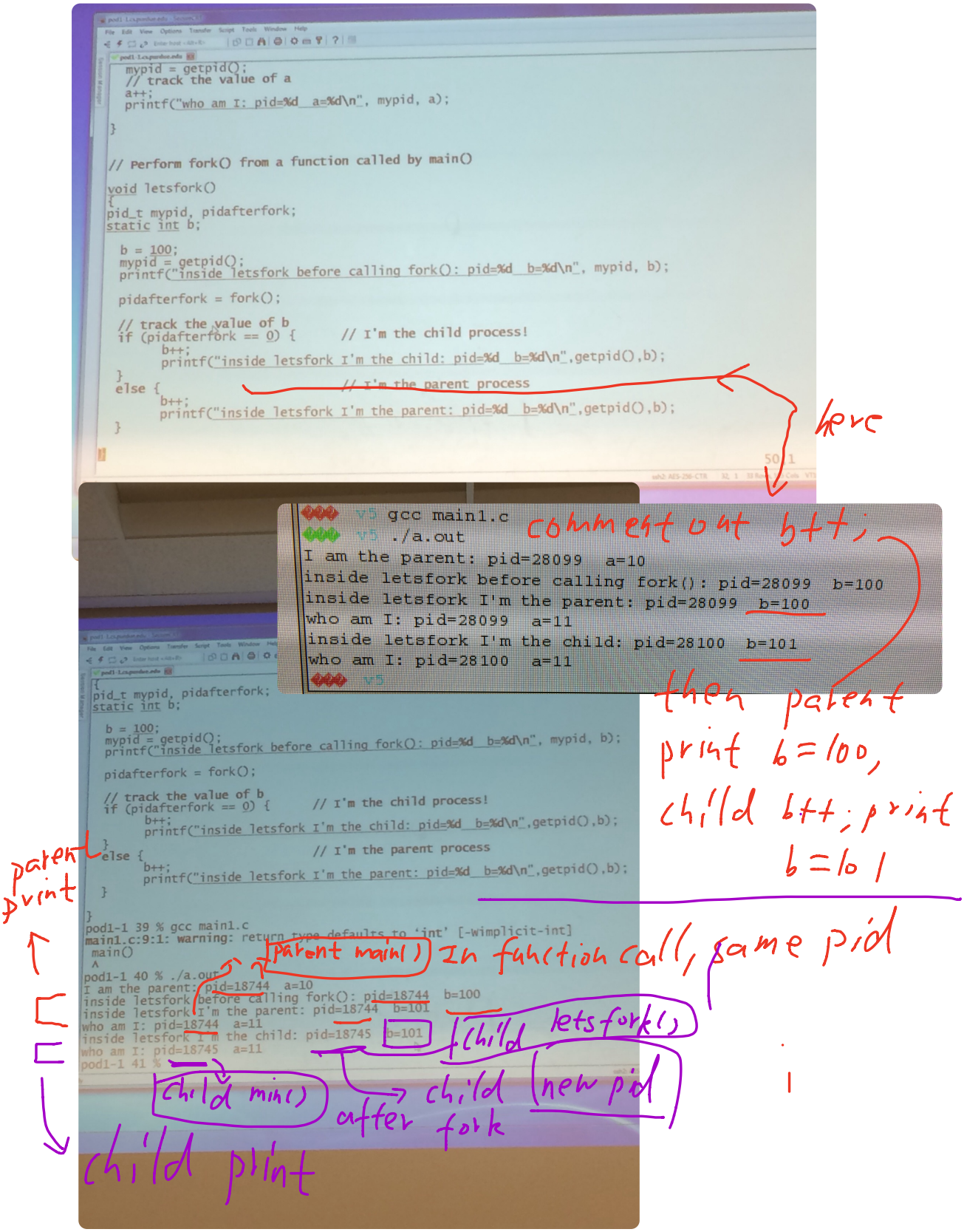
child process global int a

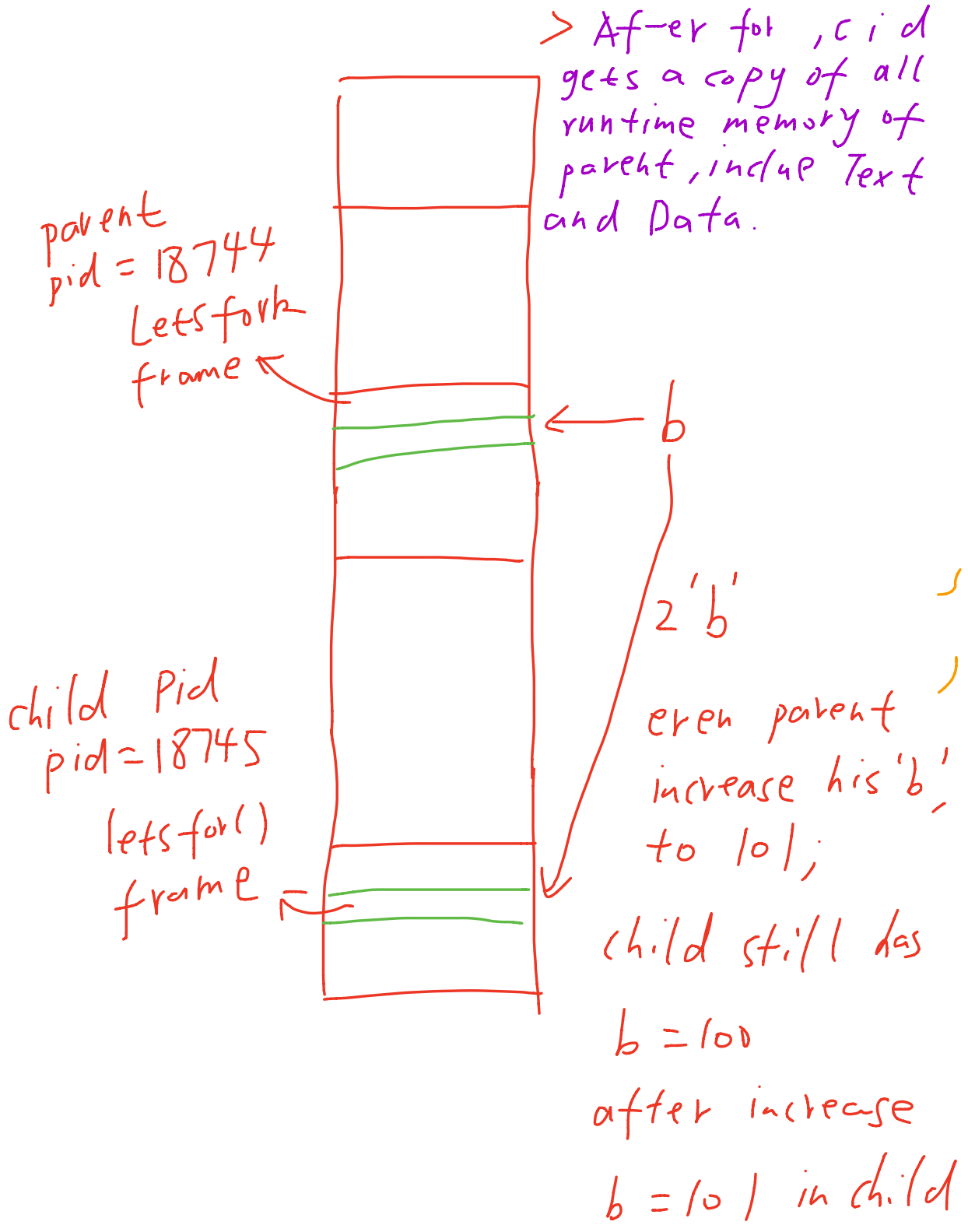
print(c_mypid, c_a);

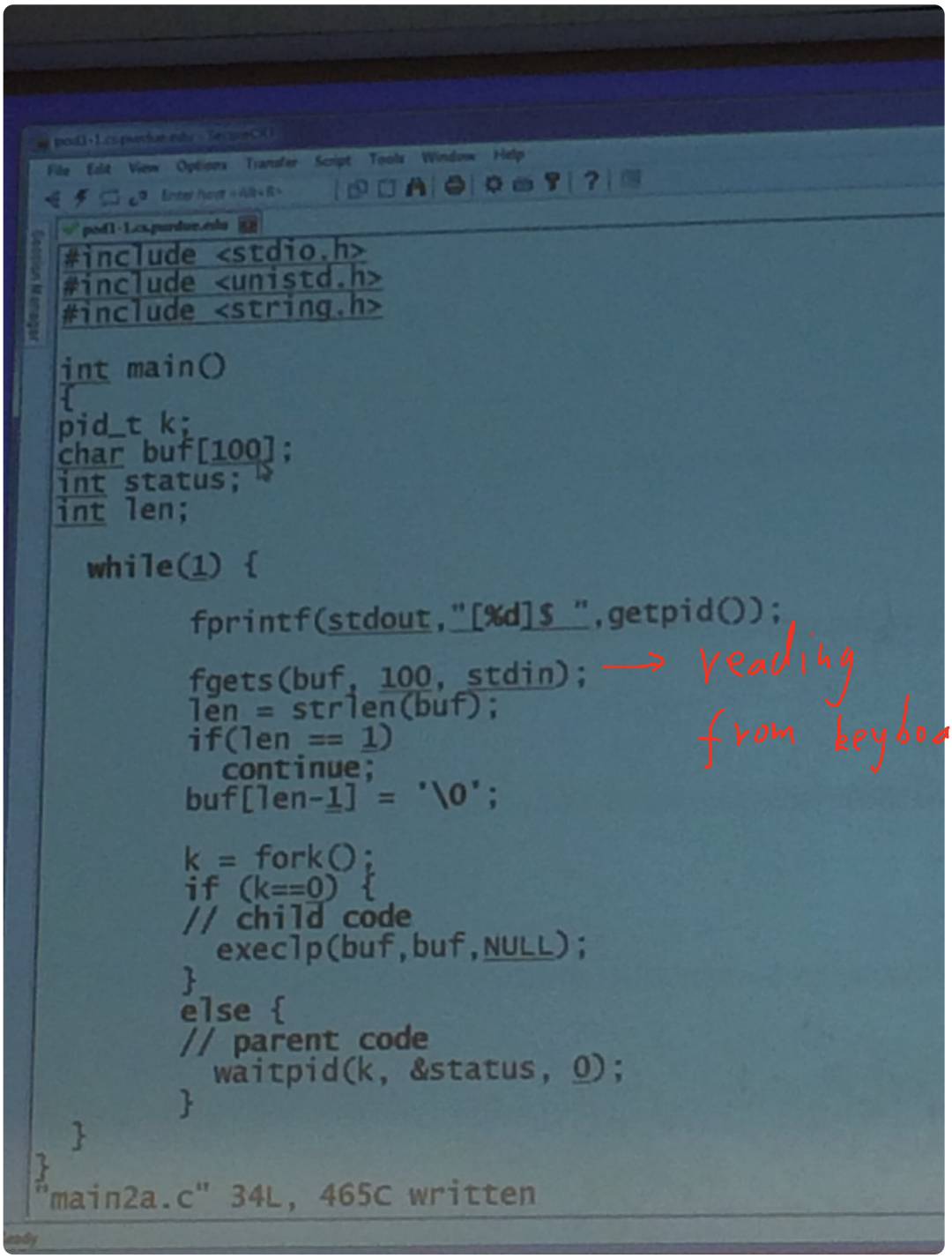
28100

101

c_a=100;
c_aff;
c_a=101;


mypid = getpid();
// track the value of a
a++;
printf("who am I: pid=%d a=%d\n", mypid, a);
}
// Perform fork() from a function called by main()
void letsfork()
{
pid_t mypid, pidafterfork;
static int b;
b = 100;
mypid = getpid();
printf("inside letsfork before calling fork(): pid=%d b=%d\n", mypid, b);
pidafterfork = fork();
// track the value of b
if (pidafterfork == 0) { // I'm the child process!
b++;
printf("inside letsfork I'm the child: pid=%d b=%d\n",getpid(),b);
}
else { // I'm the parent process
b++;
printf("inside letsfork I'm the parent: pid=%d b=%d\n",getpid(),b);
}
here
gcc main1.c
./a.out
I am the parent: pid=28099 a=10
inside letsfork before calling fork(): pid=28099 b=100
inside letsfork I'm the parent: pid=28099 b=100
who am I: pid=28099 a=11
inside letsfork I'm the child: pid=28100 b=101
who am I: pid=28100 a=11
comment out b++;
then parent print b=100,
child b++; print b=101
pod1-1 39 % gcc main1.c
main1.c:9:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^
pod1-1 40 % ./a.out
I am the parent: pid=18744 a=10
inside letsfork before calling fork(): pid=18744 b=100
inside letsfork I'm the parent: pid=18744 b=101
who am I: pid=18744 a=11
inside letsfork I'm the child: pid=18745 b=101
who am I: pid=18745 a=11
parent print
parent main()
In function call, same pid
child letsfork()
child main()
child (new pid)
after fork
child print


> After fork, child gets a copy of all runtime memory of parent, include Text and Data.
parent
pid = 18744
Lets fork
frame
b
2 'b'
even parent increase his 'b' to 101;
child Pid
pid = 18745
lets for()
frame
child still has b = 100
after increase b = 101 in child



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

int main()
{
pid_t k;
char buf[100];
int status;
int len;

  while(1) {

        fprintf(stdout,"[%d]$ ",getpid());

        fgets(buf, 100, stdin);
        len = strlen(buf);
        if(len == 1)
          continue;
        buf[len-1] = '\0';

        k = fork();
        if (k==0) {
        // child code
          execlp(buf,buf,NULL);
        }
        else {
        // parent code
          waitpid(k, &status, 0);
        }
  }
}
"main2a.c" 34L, 465C written
```

→ reading from keyboard