Solution

Problem 1: (11 points)

1 Yes. Child process created in line 8 will not be reaped by its parent process if line 9 executed before line 11, and will never be reaped by the "wait" call in line 18.

2 CBA, BCA, BAC

3 No. The parent process and child process will have duplicate but separate address spaces, so they have different copy of all variables like "c".

Problem 2: (10 points)

1 [1] IJsTU2Do2 [2] 01s

2 Since the program calls "open" without any permission bits, current user does not have read read to "c.txt", which causes the problem.

Change code in line 17 to "fd1 = open("c.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR);" or open with any mode containing read permission can fix it.

3 IIJsTU2ICIC2

Problem 3: (28 points)

1 [1] 59 [2] 2 [3] 3

2 If the high-order bits are used as an index, then some contiguous memory blocks will map to the same cache set. If the program scans an array sequentially, then the cache can only hold them in one set, cannot make full use of the whole cache. Use middle-bit indexing can map adjacent blocks to different cache sets.

3

Order	Operation	Hit/Miss	Byte Returned
1	Read 0x401368	Miss	--
2	Read 0x8036a5	Hit	0x45
3	Write 0x8036b3	Miss	--
4	Write 0x4014aa	Miss	--
5	Write 0x40139b	Hit	--
6	Read 0x8026cf	Miss	0xda

4 5

5 1. All miss.

6 Reorder line 7 and line 8.

7 ABC

Problem 4: (23 points)

1 [1] FUNCTION [2] GLOBAL [3] 00000000 [4] 4
[5] OBJ [6] G [7] .rodata [8] 4
[9] OBJ [10] L [11] .data [12]0
[13]NOTYPE [14] UND [15] 0x20(32) [16]OBJ
[17] G [18].data [19] 8 [20]COMM

2 Get XuMo (2') using x = 0 (2')

3 [1] a [2] R_X86_64_PC32 [3]draw
[4] R_X86_64_32or32S [5] a
[6] .bss [7] R_X86_64_PC32 [8] y

4 [1] af 04 20 00 [2] a2 ff ff ff
[3] e4 09 60 00 00 00 00 00
[4] fa 04 20 00 [5] d0 04 20 00

Problem 5: (7 points)

1 [1] 400440 [2] *0x6009b8 [3] 0x400456

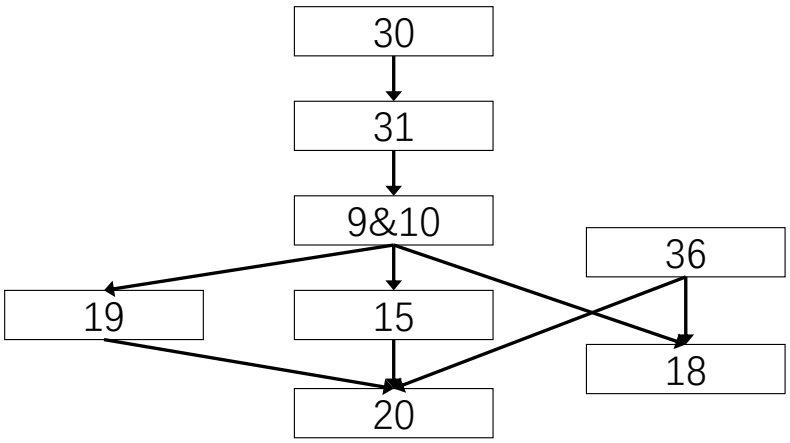
2 0x7ffff7df04e0

3 The dynamic linker determine the runtime location of atof, and overwrites GOT[3](contents in 0x6009b0) with this address.

Problem 6: (21 points)

1 [1] F [2] T [3] T [4] T [5] T [6] M
[7] ----- [8] 0/1/2/3 [9] 0
[10] 0/1/2

2



3 Sent: [1] 4 Received: [2] 3

line 30, 31, 38 and 9 could send SIGALRM to child. Though we block any signal between 30 and 31, OS may still interrupt the execution so 30 won't cancel the alarm in 30 directly.

signal from 30 and 31 will be received only once. signal from 9 will be blocked at first but will be received finally after the handle return.

4 Bug 1: If SIGCHLD is received between 40 and 41, parent will become stuck

Fix 1: change 39~41 to:

```
while (!child_exit)
    sigsuspend(&prev_one);
```

Bug 2: If all signal sent to child is received between 32 and 33, child will become stuck

Fix 2: change 32~33 to

```
sigsuspend(&prev_one);
```