# CSC209 Week 10 Notes

Hyungmo Gu

May 17, 2020

## C Pre-Processor 1 of 1

- Macros
  - Starts with '# define'
  - Can also be an expression with parameters

```
1    #define WITH_TAX(x) ((x) * 1.08) //<- NOTE: there is no space
   between WITH_TAX and (x)
2
```

    - IMPORTANT: Always surround macro variables with parenthesis

```
1    #define WITH_TAX(x) (x * 1.08)
2
3    int main() {
4        double purchase = 9.99;
5        double purchase2 = 12.49;
6
7        printf("%f\n", WITH_TAX(purchase + purchase2)); //<-
   will result in purchase + purchase2 * 1.08.
8    }
9
```

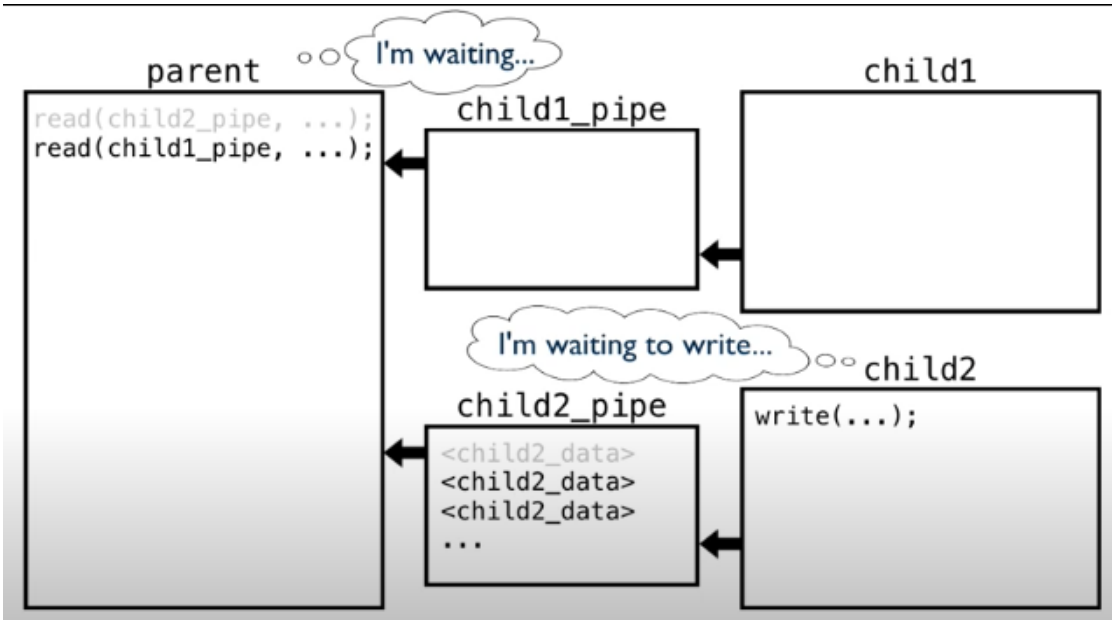
Listing 1: macros_example_1.c

## Select 1 of 2

- The problem with Blocking Reads
  - *read* waits until a pipe is non-empty, and reads one at a time
  - suppose there are multiple-children with *write*, then there may be
    1. One child with empty pipe

2. One child with filling contents, i.e. 'hello', 'hi there!'

– Parent waits for empty pipe, causing blocking




– *select* ← solution

* monitors file descriptors, waiting until one or more of the file descriptors become ready

# Select 2 of 2

- pipe
  - is a connection between two processes
  - is used for passing one process to another

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

int main()
{
    char inbuf[MSGSIZE];
    int p[2], i;

```

```c
        if (pipe(p) < 0) {
            exit(1);
        }

        /* continued */
        /* write pipe */

        write(p[1], msg1, MSGSIZE); // <- write #1
        write(p[1], msg2, MSGSIZE); // <- write #2
        write(p[1], msg3, MSGSIZE); // <- write #3

        for (i = 0; i < 3; i++) {
            /* read pipe */
            read(p[0], inbuf, MSGSIZE); // <- Read end
            printf("%s\n", inbuf);
        }
        return 0;
    }

```

Listing 2: select_example_2_1.c

- select
  - monitors file descriptors, waiting until one or more of the file descriptors become ready
  - **Syntax:** int select(numfd, read_fds, NULL, NULL, NULL);
    - **numfd:** specifies how many descriptors should be examined
    - **read_fds:** points to a bit mask that specifies the file descriptors to check for reading
    - No need to worry about NULL for now :).
  - The following macros are used
    - **FD_SET(*fd*, *&fdset*):** Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*
      - is similar to Python's set
    - **FD_ZERO (*&fdset*):** Initializes the file descriptor set *fdset* to have zero bits
    - **FD_ISSET(*fd*, *&fdset*):** Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed by *fdset*, and 0 otherwise

```c
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include <string.h>
    #include <sys/wait.h>

    #define MAXSIZE 4096
    void handle_child1(int *fd);
    void handle_child2(int *fd);

```

```
11      /* A program to illustrate the basic use of select.
12           *
13           * The parent forks two children with a pipe to read from each
    of them and then
14           * reads first from child 1 followed by a read from child 2.
15       */
16
17      int main() {
18          char line[MAXSIZE];
19          int pipe_child1[2], pipe_child2[2];
20
21          // Before we fork, create a pipe for child 1
22          if (pipe(pipe_child1) == -1) {
23              perror("pipe");
24          }
25
26          int r = fork();
27          // === line below run by parent, child 1 ===
28          if (r < 0) {
29              perror("fork");
30              exit(1);
31          } else if (r == 0) {
32              handle_child1(pipe_child1);
33              exit(0);
34          // ========================================
35          } else {
36
37              if (pipe(pipe_child2) == -1) {
38                  perror("pipe");
39              }
40
41              r = fork();
42              // === line below run by parent, child 2 ===
43              if (r < 0) {
44                  perror("fork");
45                  exit(1);
46              } else if (r == 0) {
47                  close(pipe_child1[0]);
48                  handle_child2(pipe_child2);
49                  exit(0);
50              // ====================================
51              } else {
52                  // === line run by parent ===
53                  close(pipe_child2[1]); // <- pipe closed in parent (
    since write is for children only)
54
55                  // === This part needs to be re-done each time before
    read by parent===
56                  fd_set read_fds; // <- creates set
57                  FD_ZERO(&read_fds); // <- initializes set
58                  FD_SET(pipe_child1[0], &read_fds); // <- adds
59                  FD_SET(pipe_child2[0], &read_fds);
60                  // ========================
61
```

```
62                  int numfd;
63                  if (pipe_child1[0] > pipe_child2[0]) {
64                      numfd = pipe_child1[0] + 1;
65                  } else {
66                      numfd = pipe_child2[0] + 1;
67                  }
68
69                  if (select(numfd, &read_fds, NULL, NULL, NULL) == -1)
    {
70                      perror("select");
71                      exit(1);
72                  }
73
74                  // Read first from child 1
75                  if (FD_ISSET(pipe_child1[0], &read_fds)) {
76                      if ((r = read(pipe_child1[0], line, MAXSIZE)) < 0)
     {
77                          perror("read");
78                      } else if (r == 0) {
79                          printf("pipe from child 1 is closed\n");
80                      } else {
81                          printf("Read %s from child 1\n", line);
82                      }
83                  }
84
85                  // Now read from child 2
86                  if (FD_ISSET(pipe_child2[0], &read_fds)) {
87                      if ((r = read(pipe_child2[0], line, MAXSIZE)) < 0)
     {
88                          perror("read");
89                      } else if (r == 0) {
90                          printf("pipe from child 2 is closed\n");
91                      } else {
92                          printf("Read %s from child 2\n", line);
93                      }
94                  }
95              }
96              // could close all the pipes but since program is ending
    we will just let
97              // them be closed automatically
98          }
99          return 0;
100     }
101
102     void handle_child1(int *fd) {
103         close(fd[0]);   // close read only part of pipe if children
104         printf("[%d] child\n", getpid());
105         // Child will write to parent
106         char message[10] = "HELLO DAD";
107         write(fd[1], message, 10);
108         close(fd[1]);
109     }
110
111     void handle_child2(int *fd) {
```

```
112         close(fd[0]);   // close read only part of pipe if children
113         printf("[%d] child\n", getpid());
114         // Child will write to parent
115         char message[10] = "Hi mom";
116         write(fd[1], message, 10);
117         close(fd[1]);
118     }
119
```

Listing 3: select_example_2_1.c