9.5/10

COSC 350 System Software (Mini Test #2)

9/29/21

Name: Jung An

1. (4 pt.) Write a compliable C program named "**singledigitsumup.c**" that reads sequence of positive or negative integers on the command line and prints their sum of single digit numbers.
   Exit the program with an appropriate error message under the following error condition: if there is no any integer on the command line.

   Ex) for ./singledigitsumup -5 9 11 8 100 -4 7 22
   Output: The sum of single argument is 15
   You need define a function (named **st_to_int**) to change a positive and negative numerical c-string to integer. (Do not use atoi() or any library functions)

```
#include <stdio.h>
#include <stdlib.h>

int st_to_int(char *str)
{
    int i=0;
    int negative=0;
    int number=0;
    int stringCount=0;
    if(str[0]=='-')
    negative=1;
    for(i=0; str[i]!='\0'; i++)
        stringCount++;
    number = str[stringCount]-'0';

    if(negative==1)
        return -1*number;
    return number;
}

int main(int argc, char** argv)
{
    if(argc==1)
    {
        printf("No integer input");
        return 1;
    }
    int Sum=0;
    int i;
    for(i=1; i<=argc; i++)
        Sum+=st_to_int(argv[i]);
    printf("The Sum of single argument is %i", Sum);
    return 0;
}
```

good!

you must calculate single digit only!

```
#include <stdio.h>
#include <stdlib.h>
```

2. (4 pt.) Write a compliable C program which open a file (**foo**) as a input and write into a file (**mirror**) as a mirror of **foo** (created output file mode will be rw-------)by using the lseek system call.
   Ex) If content of foo is a string **abcdefg**, content of mirror will be **gfedcba || abcdefg**.

```
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>

#define BUFFER_SIZE 1
int main()
{  char buffer[BUFFER_SIZE];
   int nread;
   int f_in, f_out;

   if ((f_in = open("foo", O_RDONLY)) == -1)
   {  write(2, "Open error!\n", 12);
      return 1;
   }
   umask(0000);
   if ((f_out = open("mirror", O_WRONLY | O_CREAT, 0600)) == -1)
   {write(2, "Open error!\n", 12);
      return 1;
   }
   int offset = lseek(f_in, 0, SEEK_END);

   while (offset >0)
   {   if ((nread = read(f_in, buffer, BUFFER_SIZE)) == -1)
       {  write(2, "Read error!\n", 12);
          return 2;}
       if (write(f_out, buffer, nread) != nread)
       {  write(2, "write error!\n", 13);
          return 3;}
       lseek(f_in, -2, SEEK_CUR);
       offset--;
   }
  char mirror[4] = " || ";  // there is a space at the beginning and end.
  if (write(f_out, mirror, 4) == -1)
  {write(2, "write error!\n", 13);
     return 3;}
  lseek(f_in, 0, SEEK_SET);

  while ((nread = read(f_in, buffer, BUFFER_SIZE))>0)
  {  if (write(f_out, buffer, nread) != nread)
     {write(2, "write error!\n", 13);
        return 3;
     }
  }
  if (nread == -1){
   write(2, "Read error!\n", 12);
   return 2;
  }
  close(f_in);
  close(f_out);
  return 0;
}
```

Good!

3. (1 pt.)Probably the most important issue in implementing file storage is keeping track of which disk blocks go with which file. Various methods are used in different operating systems: contiguous allocation, linked-list allocation, linked-list with FAT and index-node. Briefly and clearly discuss each idea.

   - Contiguous allocation – The blocks are continuous Next to each other ✓
   - Linked-list allocation – The block contains the value and the next linked block. ✓
   - Linked-list allocation with FAT(file allocation table) – The linked block are saved in FAT so all the linked blocks are shown ✓
   - Index-Node – The block information is saved in i-node (index node) ✓

4. (1 pt.) Write a compliable C program that reads a file name on the command line and prints the file size by bytes. Do not use lseek(), pread(), pwrite() stat(), fstat() or lstat() system call.

```
#include <stdio.h>
#include<stdlib.h>
#include<fcntl.h>

#define BUFFER_SIZE 1
int main (int argc, char **argv)
{
    if (argc != 2)
    { printf("Need one file name");
      return 1;
    }
    char * f_iName = argv[1];
    int f_in;

    if (f_in = open (f_iName, O_RDONLY) == -1)
    { printf ("Open error!\n");
      return 2;
    }
    char buffer[BUFFER_SIZE];
    int nread;
    int byte=0;
    while((nread= read (f_in, buffer, BUFFER_SIZE)) >0)
          byte += nread;

       if (nread == -1)
       { printf ("Read error!\n");
         return 3;
       }
       printf (" File size in bytes: %i", byte);
       close(f_in);
       return 0;
}
```

← This requires #include <unistd.h> library. Could not figure out how to do this without this library