Assignment3

Team18

112062519 廖思愷
112062636 游竣量
111065547 游述宇

- Describe the implementation:
  1. 標頭檔

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

引入必要的標頭文件。
stdio.h 用於 I/O 操作。
string.h 提供字符串和記憶體操作函數。
stdlib.h 用於分配和釋放記憶體。

  2. 常數定義

```c
#define BUFFERSIZE 20
#define MEMORYSIZE 20
```

定義兩個常數，BUFFERSIZE 是緩衝區大小，MEMORYSIZE 是模擬的記憶體大小。

  3. Memory 結構

```c
typedef struct Memory{
    char* arr;
    int size;
    int pos;
}Memory;
```

這是 struct 用於表示模擬的記憶體。其中：

arr 是一個 char pointer，指向記憶體的首地址。
size 表示記憶體的大小。
pos 表示當前的讀寫位置。

4. readfn( )

```
int readfn(void* ptr, char* buf, int words){
    Memory* mem = (Memory*)ptr;
    int return_value = 0;
    int words_to_read = 0;
    int remain_space = mem->size - mem->pos;

    if (remain_space >= words) words_to_read = words;
    else words_to_read = remain_space;

    memcpy(buf, mem->arr + mem->pos, words_to_read);
    mem->pos += words_to_read;

    return words_to_read;
}
```

功能：為從記憶體中讀取數據。
其中：

ptr 指向 Memory 結構。
buf 是一個字符緩衝區，用於存放從記憶體中讀取的數據。
words 表示想要讀取的字符數。

這個 function 首先計算從當前位置到記憶體結尾的剩餘空間。然後，基於所需讀取的字符數和剩餘空間，確定實際要讀取的字符數。之後使用 memcpy 從 Memory 結構的 arr 中讀取數據到緩衝區。最後更新 pos 並返回讀取的字符數。

5. writefn( )

```
int writefn(void* ptr, const char* buf, int words){
    Memory* mem = (Memory*)ptr;
    int return_value = 0;
    int words_to_write = 0;
    int remain_space = mem->size - mem->pos;

    if (remain_space >= words) words_to_write = words;
    else words_to_write = remain_space;

    memcpy(mem->arr + mem->pos, buf, words_to_write);
    mem->pos += words_to_write;

    return words_to_write;
}
```

功能：將數據從給定的緩衝區寫入到記憶體中。

同樣的，首先計算從當前位置到結尾的剩餘空間。
根據所需寫入的字符數和剩餘空間，確定實際要寫入的字符數。
使用 memcpy 將數據從緩衝區寫入到 Memory 結構的 arr。
更新 pos 並返回寫入的字符數。

6. seekfn( )

```c
fpos_t seekfn(void* ptr, fpos_t offset, int whence){
    Memory* mem = (Memory*)ptr;
    switch (whence) {
        case SEEK_SET:
            mem->pos = offset;
            break;
    }

    return mem->pos;
}
```

功能：用於修改當前的讀寫位置。

基於 whence 的值（只考慮 SEEK_SET），設置 pos 的值。如果是 SEEK_SET，則將 pos 設置為 offset。
返回新的 pos 位置。

7. closefn( )

```c
int closefn(void* ptr){
    free(ptr);
    return 0;
}
```

功能：當文件流被關閉時調用的。它釋放了為 Memory 結構分配的記憶體，並返回 0 表示成功。

8. my_fmemopen( )

```c
FILE* my_fmemopen(int size){
    Memory* mem = (Memory*)calloc(1, sizeof(Memory));
    mem->arr = calloc(size, sizeof(char));
    mem->size = size;
    mem->pos = 0;
    FILE* stream = funopen(mem, readfn, writefn, seekfn, closefn);
    return stream;
}
```

功能：創建一個基於記憶體的文件流。

首先，為 Memory 結構分配記憶體，並為其 arr 成員分配指定大小的記憶體。
設置 size 和 pos 成員的值。
使用 funopen 函式，以 Memory 結構和先前定義的回調函式（readfn, writefn, seekfn, closefn）為參數，創建一個文件流。
返回這個新創建的文件流。

9. main( )

```
char read_buf[BUFFERSIZE] = {0};
char write_buf[BUFFERSIZE] = "hello, world";
```

read_buf：用於讀取從記憶體文件流中的數據。
write_buf：已初始化為 "hello, world"，稍後將其寫入記憶體文件流。

```
int size = MEMORYSIZE;
```

定義一個名為 size 的整數，並初始化為 MEMORYSIZE（定義為 20）。

```
FILE* stream = my_fmemopen(size);
```

使用先前定義的 my_fmemopen 函式，創建一個大小為 size 的記憶體文件流。這裡的 stream 是一個指向這個記憶體文件流的指針。

```
setvbuf(stream, NULL, _IONBF, 0);
```

使用 setvbuf 函式設置 stream 為無緩衝模式。表示對 stream 的所有讀/寫操作都會立即影響到底層的記憶體，而不是先被存放到一個緩衝區中。
如果沒有設置成無緩衝模式， readfn 的 words 會設定為 1024，造成讀取錯誤。

```
fwrite(write_buf, 1, 12, stream);
```

使用 fwrite 函式（實際會呼叫我定義的 writefn）將 write_buf 中的前 12 個字符寫入到 stream 中。這會將 "hello, world" 寫入記憶體文件流。

```
fseek(stream, 7, SEEK_SET);
```
使用 fseek 函式（實際會呼叫我定義的 seekfn）將 stream 的文件指針移動到第 7 個字符位置。考慮到 "hello, world" 的內容，這將指針移動到 "w" 的位置。

```
fread(read_buf, 1, 5, stream);
```
使用 fread 函式（實際會呼叫我定義的 readfn）從 stream 讀取 5 個字符到 read_buf。執行完後 read_buf 將包含 "world"。

```
read_buf[5] = '\0';
```
在 read_buf 中添加一個結束字符，確保它是一個有效的 C 字符串。

```
printf("%s\n", read_buf);
```
print read_buf 的內容。輸出將是 "world"。

```
fseek(stream, 0, SEEK_SET);
```
再次使用 fseek （實際會呼叫我定義的 seekfn）將 stream 的文件指針重置回開始位置。

```
fread(read_buf, 1, 12, stream);
```
從 stream 讀取 12 個字符到 read_buf。

```
read_buf[12] = '\0';
```
再次在 read_buf 中添加一個結束字符。

```
printf("%s\n", read_buf);
```
print read_buf 的內容。輸出將是 "hello, world"。

```
fclose(stream);
```
關閉 stream。這也會呼叫 closefn，從而釋放相關的記憶體。

- screenshot of result