Problem 1:

Write a version of the **`fgets`** function, called **`tfgets`**, that times out after 5 seconds. The **`tfgets`** function accepts the same inputs as **`fgets`**. If the user doesn't type an input line within 5 seconds, **`tfgets`** returns NULL. Otherwise, it returns a pointer to the input line.

```
Char *tfgets(char* str, int num, FILE* stream) {
	Int current = 0;
	Int end;
	Int i = 0;

	if(str == NULL | stream == NULL | num <= 0) {
		Return NULL;
	}

	char* out = (char*)malloc(sizeof(char) * n);
	while(num){
		Start = clock();
		End = start + 5000;

		while(current < end){
			current =  clock();
		}

		if(current > end){
			Break;
		}

		Char c = fgetc(stream);
		if(c != '\n' && c != EOF) {
			Out[i] = c;
		} else {
			Break;
		}
		i++;
	}

	Out[i] = '\0';

	 if(i != 0) {
		Str = out;
		Return str;
	}
}
```

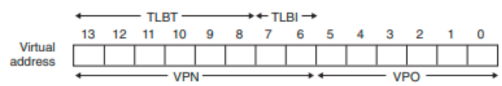
Problem 2:

In the following problem, you are to show how the given memory system example translates a virtual address into a physical address and accesses the cache. For each virtual address, indicate the virtual page number (VPN) accessed, the TLB entry accessed, the physical address, and the cache byte value returned. Indicate whether the TLB misses, whether a page fault occurs, and whether a cache miss occurs. If there is a cache miss, enter "—" for "Cache byte value" If there is a page fault, enter "--" for "PPN" and leave the cells from PA to cache byte value blank.

VA = 0x027c **0000001001111100**	**VA = 0x0915** **00100100010101**	**VA = 0x0040** **00000001000000**
VPN = 00000010 => 0x02	VPN = **00100100 => 0x24**	VPN = **00000001 => 0x01**
TLB index = 10 => 2	TLB index = 00 => 0	TLB index = 01 => 1
TLB tag = 000000 => 0x00	TLB tag = 001001 => 0x09	TLB tag = 000000 => 0x00
TLB hit? (Y/N) N	TLB hit? (Y/N) Y	TLB hit? (Y/N) N
Page fault? (Y/N) N	Page fault? (Y/N) N	Page fault? (Y/N) N
PPN = 0x33	PPN = 0x0d	PPN = -
PA = 110011 **111100**	PA = PPN + PPO 001101 **010101**	PA = 000000 **000000**
CO = 00 => 0	CO = 01 => 1	CO = 00 => 0
CI = 1111 => 15 = F	CI = **0101 => 5**	CI = 0000 => 0

CT = 0x33	CT = 0x0d	CT = 0x00
Cache hit? (Y/N) N	Cache hit? (Y/N) Y	Cache hit? (Y/N) N
Cache byte value	Cache byte value 0x36	Cache byte value

Example Memory System:

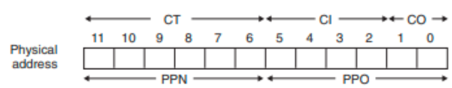


Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

(a) TLB: 4 sets, 16 entries, 4-way set associative

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	—	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	—	0
04	—	0	0C	—	0
05	16	1	0D	2D	1
06	—	0	0E	11	1
07	—	0	0F	0D	1

(b) Page table: Only the first 16 PTEs are shown



Idx	Tag	Valid	Blk 0	Blk 1	Blk 2	Blk 3
0	19	1	99	11	23	11
1	15	0	—	—	—	—
2	1B	1	00	02	04	08
3	36	0	—	—	—	—
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	—	—	—	—
7	16	1	11	C2	DF	03
8	24	1	3A	00	51	89
9	2D	0	—	—	—	—
A	2D	1	93	15	DA	3B
B	0B	0	—	—	—	—
C	12	0	—	—	—	—
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	—	—	—	—

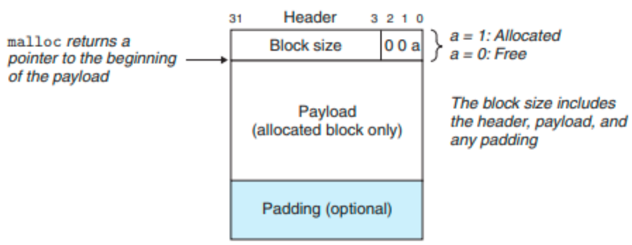
(c) Cache: 16 sets, 4-byte blocks, direct mapped

Problem 3:

Determine the block sizes and header values that would result from the following sequence of malloc requests. Assumptions: (1) The allocator maintains double-word alignment and uses an implicit free list with the block format from Figure 9.35. (2) Block sizes are rounded up to the nearest multiple of 8 bytes.

Request	Block size (decimal)	Block header (hex)
`malloc(4)`	4+4 = 8 bytes	1001 = 0x09
`malloc(7)`	7 + 4 = 11 => 16 bytes	00010001 = 0x11
`malloc(19)`	19 + 4 = 23 => 24 bytes	00011001 = 0x19
`malloc(22)`	22 + 4 = 26 => 32 bytes	00010101 = 0x21

Figure 9.35
Format of a simple heap block.



Problem 4:

Determine the minimum block size for each of the following combinations of alignment requirements and block formats. Assumptions: Explicit free list, 4-byte pred and succ pointers in each free block, zero-size payloads are not allowed, and headers and footers are stored in 4-byte words.

Alignment	Allocated block	Free block	Minimum block size (bytes)
Single Word	Header and footer	Header and Footer	16 bytes
Single Word	Header only	Header and Footer	20 bytes
Double Word	Header and footer	Header and Footer	24 bytes
Double Word	Header only	Header and Footer	28 bytes