CS 332/532 Systems Programming

Lecture 15

-Standard I/O Libraries-

Professor : Mahmut Unan – UAB CS

Agenda

- Standard I/O continued
- strtok
- atoi
- atof
- strdup
- Process management

Example 2

- We will now write a program to read a comma separated file (“listing.csv”) and use the C structures to store and display the data on the console.
- The sample input file used is :

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	id	host_id	host_name	neighbourho	neighbourho	latitude	longitude	room_type	price	minimum_ni	number_of_	calculated_h	availability_365
2	2015	2217	Ian	Mitte	Brunnenstr. S	52.5345373	13.4025569	Entire home/	60	4	118	4	141
3	2695	2986	Michael	Pankow	Prenzlauer B	52.5485128	13.4045528	Private room	17	2	6	1	0
4	3176	3718	Britta	Pankow	Prenzlauer B	52.5349962	13.4175787	Entire home/	90	62	143	1	220
5	3309	4108	Jana	Tempelhof -	SchVðneberg	52.4988549	13.3490645	Private room	26	5	25	1	297
6	7071	17391	Bright	Pankow	Helmholtzpla	52.5431573	13.4150911	Private room	42	2	197	1	26
7	9991	33852	Philipp	Pankow	Prenzlauer B	52.5330308	13.4160468	Entire home/	180	6	6	1	137
8	14325	55531	Chris + Olive	Pankow	Prenzlauer B	52.5478464	13.4055622	Entire home/	70	90	23	3	129
9	16401	59666	Melanie	Friedrichshai	Frankfurter A	52.510514	13.4578502	Private room	120	30	0	1	365
10	16644	64696	Rene	Friedrichshai	nVðrdliche Lu	52.5047923	13.4351019	Entire home/	90	60	48	2	159
11	17409	67590	Wolfram	Pankow	Prenzlauer B	52.5290709	13.4128434	Private room	45	3	279	1	42
12	17904	68997	Matthias	NeukVðlln	ReuterstraVü	52.4954763	13.4218213	Entire home/	49	5	223	1	232
13	20858	71331	Marc	Pankow	Prenzlauer B	52.5369524	13.407615	Entire home/	129	3	56	1	166
14	21869	64696	Rene	Friedrichshai	nVðrdliche Lu	52.5027333	13.4346199	Entire home/	70	60	60	2	129
15	22415	86068	Kiki	Friedrichshai	sVºdliche Lui	52.4948506	13.4285006	Entire home/	98	3	61	2	257
16	22677	87357	Ramfis	Mitte	Brunnenstr. S	52.5343484	13.4055765	Entire home/	160	3	223	1	228
17	23834	94918	Tanja	Friedrichshai	Tempelhofer	52.4897144	13.3797476	Entire home/	65	60	96	1	275
18	24569	99662	Dominik	Pankow	Prenzlauer B	52.5307909	13.4180844	Entire home/	90	3	18	2	3
19	25653	99662	Dominik	Pankow	Prenzlauer B	52.5302587	13.419467	Entire home/	90	4	5	2	15
20	26543	112675	Terri	Pankow	Helmholtzpla	52.5440624	13.4213765	Entire home/	197	3	163	1	336
21	28156	55531	Chris + Olive	Pankow	Prenzlauer B	52.5467194	13.405117	Entire home/	70	90	28	3	191
22	28268	121580	Elena	Friedrichshai	Frankfurter A	52.5133852	13.4699475	Entire home/	90	5	30	1	55
23	28711	84157	Emanuela	NeukVðlln	ReuterstraVü	52.4861061	13.434817	Entire home/	60	2	1	10	341
24	29279	54283	Marine	Pankow	Helmholtzpla	52.5417876	13.4238832	Entire home/	130	60	69	3	221

listing.c

- The given file has 13 different attributes, and these attributes can be divided into three different datatypes: integer, character array, and float.
- And collectively we can create a C structure to represent these attributes and then create an array of such structures to store multiple entities.
- 1. Define a structure called listing with all attributes as individual members of the struct listing.

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

#define LINESIZE 1024

struct listing {
    int id, host_id, minimum_nights, number_of_reviews, calculated_host_listings_count,availability_365;
    char *host_name, *neighbourhood_group, *neighbourhood, *room_type;
    float latitude, longitude, price;
};
```

strtok

- `char *strtok(char *str, const char *delim);`
- `char *strtok_r(char *str, const char *delim, char **saveptr);`

- strtok, strtok_r - extract tokens from strings

- The strtok() function breaks a string into a sequence of zero or more nonempty tokens. On the first call to strtok(), the string to be parsed should be specified in *str*. In each subsequent call that should parse the same string, *str* must be NULL.

https://man7.org/linux/man-pages/man3/strtok.3.html

atoi

- atoi, atol, atoll - convert a string to an integer

- `int atoi(const char *nptr);`
- `long atol(const char *nptr);`
- `long long atoll(const char *nptr);`

- Return value: The converted value or 0 on error.

https://man7.org/linux/man-pages/man3/atoi.3.html

atof

- atof - convert a string to a double

```
double atof(const char *nptr);
```

RETURN VALUE : The converted value

https://man7.org/linux/man-pages/man3/atof.3.html

strdup

- strdup- duplicate a string

```
char *strdup(const char *s);
```

On success, the strdup() function returns a pointer to the duplicated string. It returns NULL if insufficient memory was available, with errno set to indicate the cause of the error.

https://www.man7.org/linux/man-pages/man3/strndup.3.html#:~:text=DESCRIPTION%20top,copies%20at%20most%20n%20bytes.

listing.c

- 2. Define a function which can help to parse each line in the file and return the above defined structure.
- For this task you need to learn the string tokenizer function (*strtok*) which is available in <*string.h*> header file.
- You can find out more about the *strtok* function by typing *man strtok*.
- You will notice that when you invoke the *strtok* function for the first time you provide the pointer to the character array and on subsequent invocations of *strtok* we use *NULL* as the argument.

listing.c

```
15  struct listing getfields(char* line){
16      struct listing item;
17
18      item.id = atoi(strtok(line, ","));
19      item.host_id = atoi(strtok(NULL, ","));
20      item.host_name = strdup(strtok(NULL, ","));
21      item.neighbourhood_group = strdup(strtok(NULL, ","));
22      item.neighbourhood = strdup(strtok(NULL, ","));
23      item.latitude = atof(strtok(NULL, ","));
24      item.longitude = atof(strtok(NULL, ","));
25      item.room_type = strdup(strtok(NULL, ","));
26      item.price = atof(strtok(NULL, ","));
27      item.minimum_nights = atoi(strtok(NULL, ","));
28      item.number_of_reviews = atoi(strtok(NULL, ","));
29      item.calculated_host_listings_count = atoi(strtok(NULL, ","));
30      item.availability_365 = atoi(strtok(NULL, ","));
31
32      return item;
33  }
```

listing.c

Now, create a function to display the items in the struct

```
36  void displayStruct(struct listing item) {
37      printf("ID : %d\n", item.id);
38      printf("Host ID : %d\n", item.host_id);
39      printf("Host Name : %s\n", item.host_name);
40      printf("Neighbourhood Group : %s\n", item.neighbourhood_group);
41      printf("Neighbourhood : %s\n", item.neighbourhood);
42      printf("Latitude : %f\n", item.latitude);
43      printf("Longitude : %f\n", item.longitude);
44      printf("Room Type : %s\n", item.room_type);
45      printf("Price : %f\n", item.price);
46      printf("Minimum Nights : %d\n", item.minimum_nights);
47      printf("Number of Reviews : %d\n", item.number_of_reviews);
48      printf("Calculated Host Listings Count : %d\n", item.calculated_host_listings_count);
49      printf("Availability_365 : %d\n\n", item.availability_365);
50  }
```

listing.c

- 3. Now use *fopen* function to open file in read only mode. Notice that the *fopen* function returns a pointer of *FILE* type (file pointer) unlike the *open* function that returns an *integer* value as the file descriptor.

```
52  int main(int argc, char* args[]) {
53      struct listing list_items[22555];
54      char line[LINESIZE];
55      int i, count;
56
57      FILE *fptr = fopen("listings.csv", "r");
58      if(fptr == NULL){
59          printf("Error reading input file listings.csv\n");
60          exit (-1);
61      }
```

listing.c

- 4. Then loop through till the end of file (use fgets function) and store all data in the array of structures
- We can close the file using fclose function

```
62
63      count = 0;
64      while (fgets(line, LINESIZE, fptr) != NULL){
65          list_items[count++] = getfields(line);
66      }
67      fclose(fptr);
```

listing.c

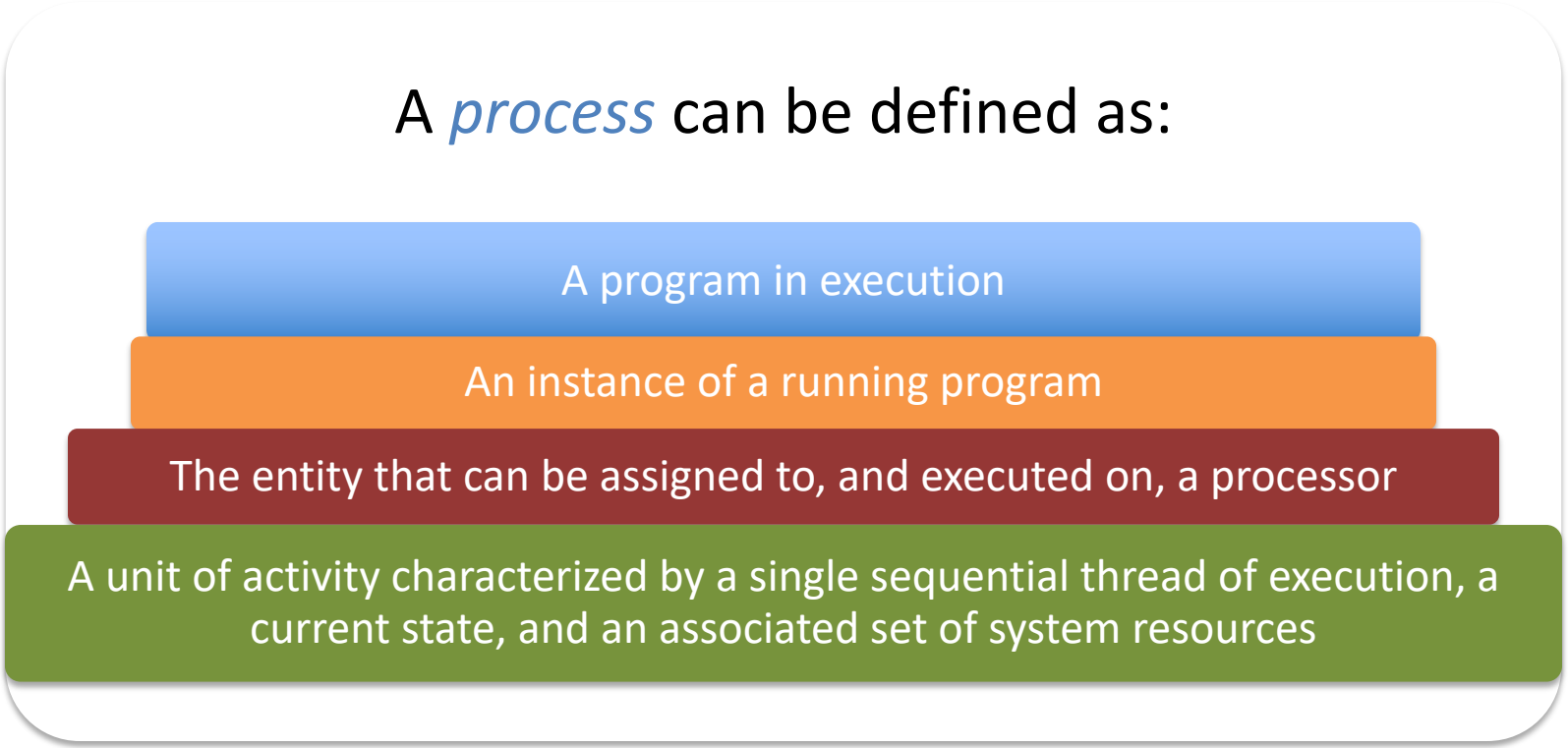
- 5. Now invoke the function to display the structure in a loop.

```
69      for (i=0; i<count; i++)
70          displayStruct(list_items[i]);
71
72      return 0;
73  }
74
```

Process Management

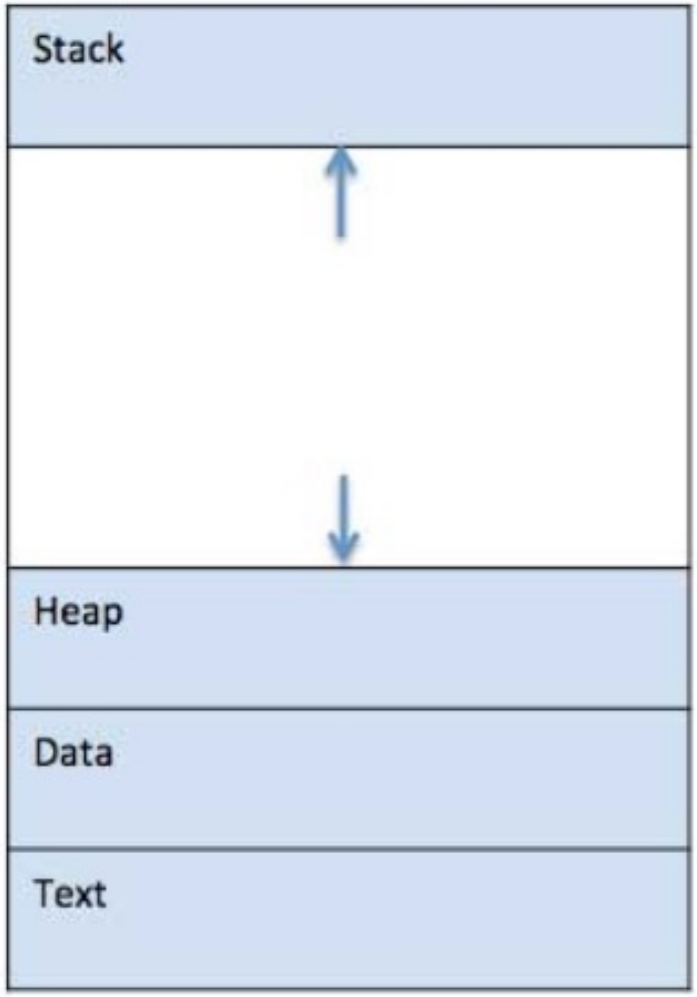
Process

- Fundamental to the structure of operating systems



Process

- When a program is loaded into the memory and it becomes a process, it can be divided into four sections
  - stack
  - heap
  - text
  - data.

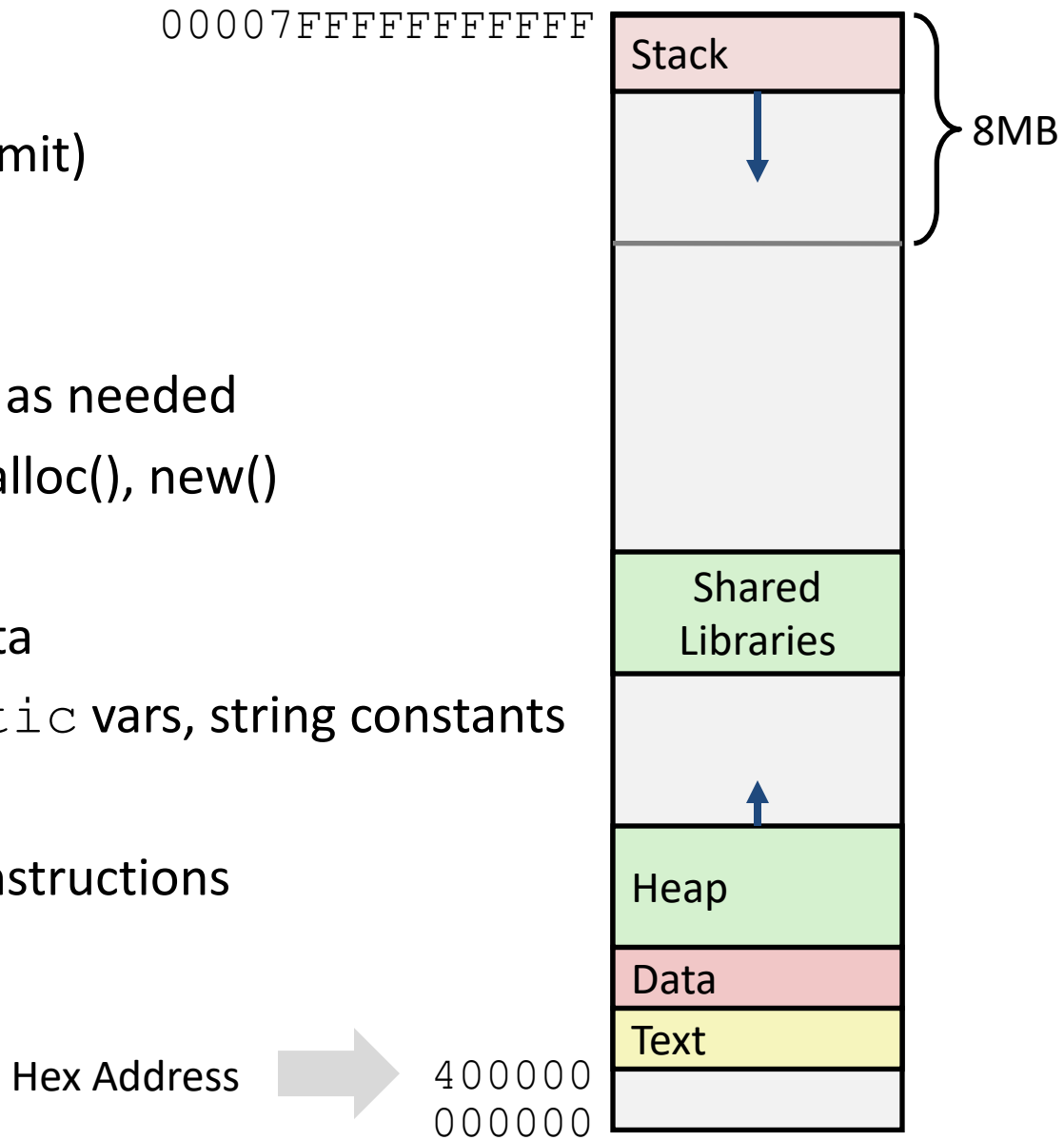


https://www.tutorialspoint.com/operating_system/os_processes.htm

x86-64 Linux Memory Layout

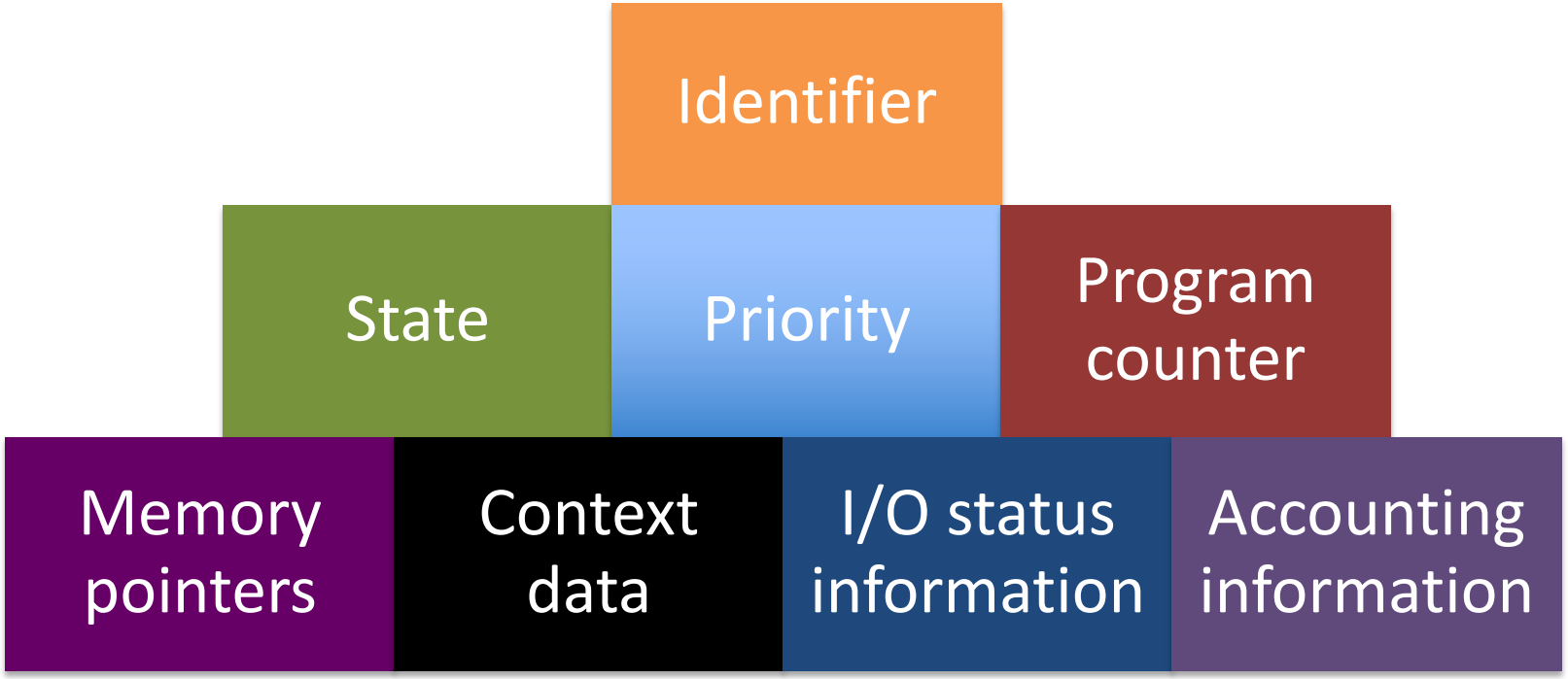
not drawn to scale

- Stack
  - Runtime stack (8MB limit)
  - E. g., local variables
- Heap
  - Dynamically allocated as needed
  - When call malloc(), calloc(), new()
- Data
  - Statically allocated data
  - E.g., global vars, `static` vars, string constants
- Text / Shared Libraries
  - Executable machine instructions
  - Read-only



Process Elements

- While the program is executing, this process can be uniquely characterized by a number of elements, including:



Process Execution

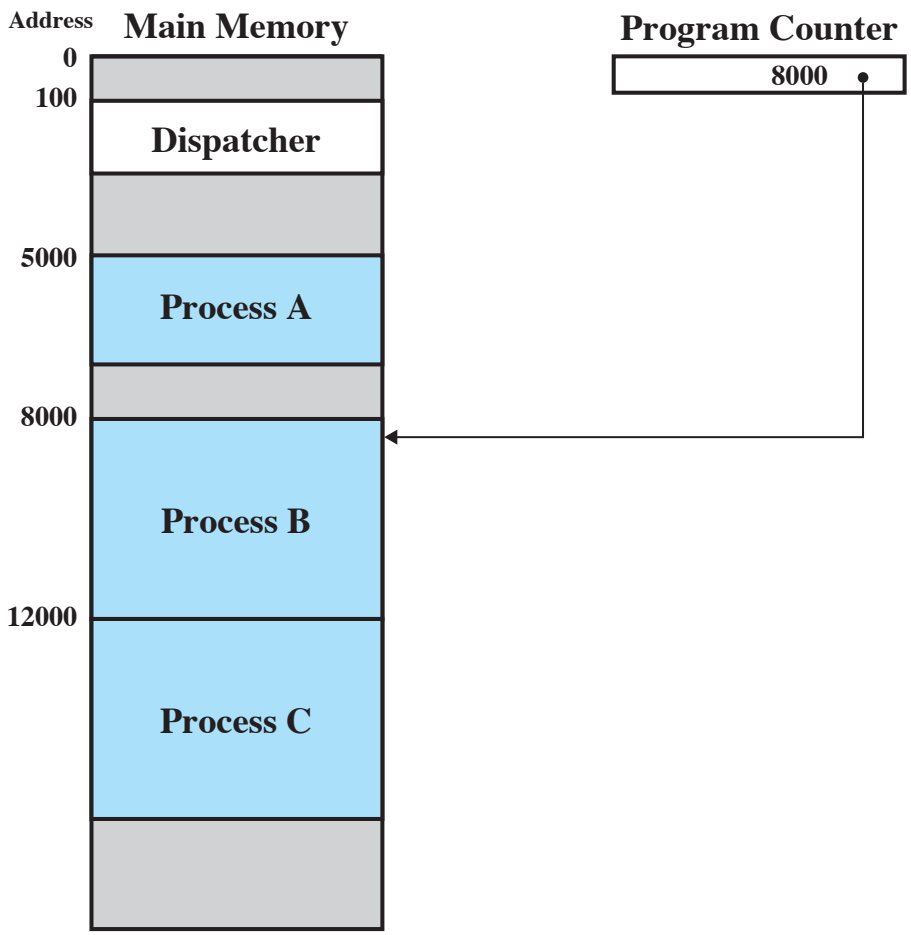


Figure 3.2 Snapshot of Example Execution (Figure 3.4) at Instruction Cycle 13

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

(a) Trace of Process A **(b) Trace of Process B** **(c) Trace of Process C**

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

Figure 3.3 Traces of Processes of Figure 3.2

1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
----------	Timeout
7	100
8	101
9	102
10	103
11	104
12	105
13	8000
14	8001
15	8002
16	8003
----------	I/O Request
17	100
18	101
19	102
20	103
21	104
22	105
23	12000
24	12001
25	12002
26	12003
27	12004
28	12005
----------	Timeout
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
----------	Timeout
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011
----------	Timeout

100 = Starting address of dispatcher program

Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Figure 3.4 Combined Trace of Processes of Figure 3.2

Process Creation

Process spawning

- When the OS creates a process at the explicit request of another process

Parent process

- Is the original, creating, process

Child process

- Is the new process

Process Termination

- There must be a means for a process to indicate its completion
- A batch job should include a HALT instruction or an explicit OS service call for termination
- For an interactive application, the action of the user will indicate when the process is completed (e.g. log off, quitting an application)

Five-State Process Model

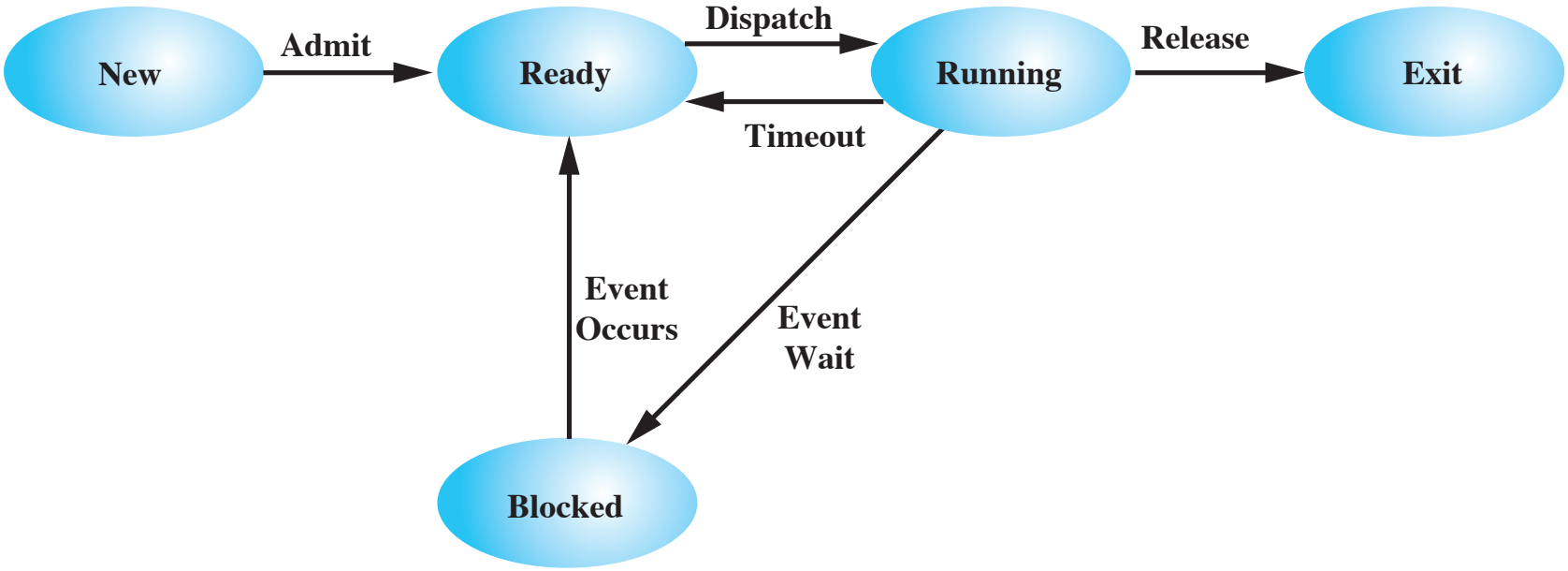
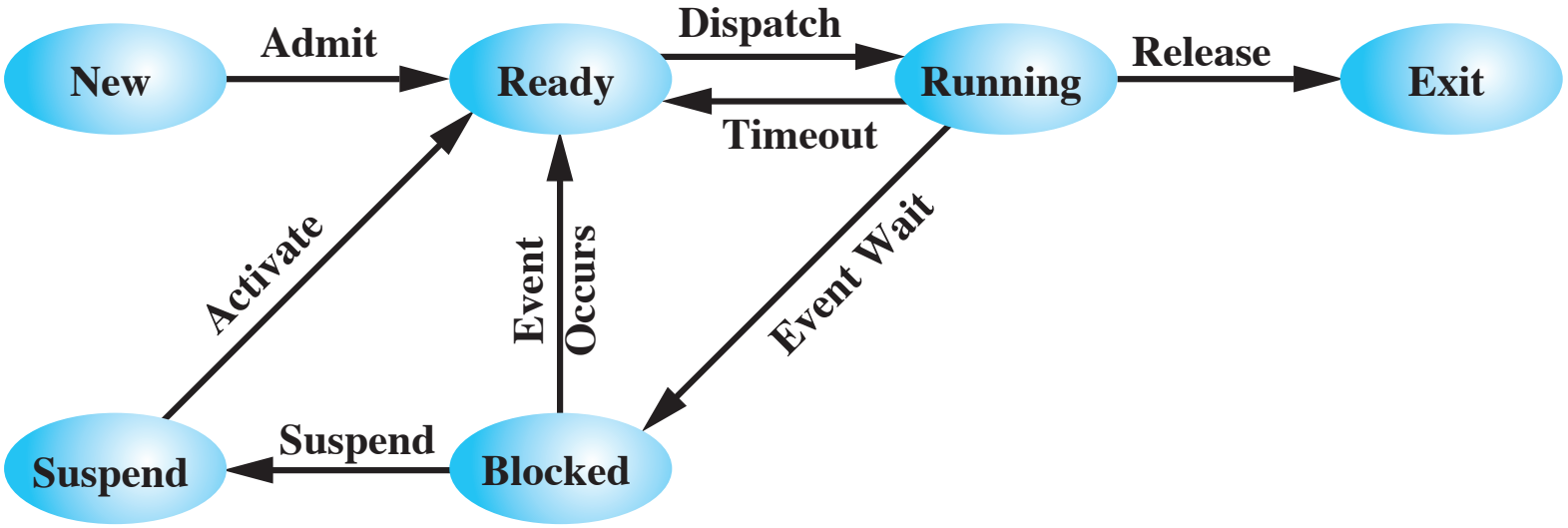
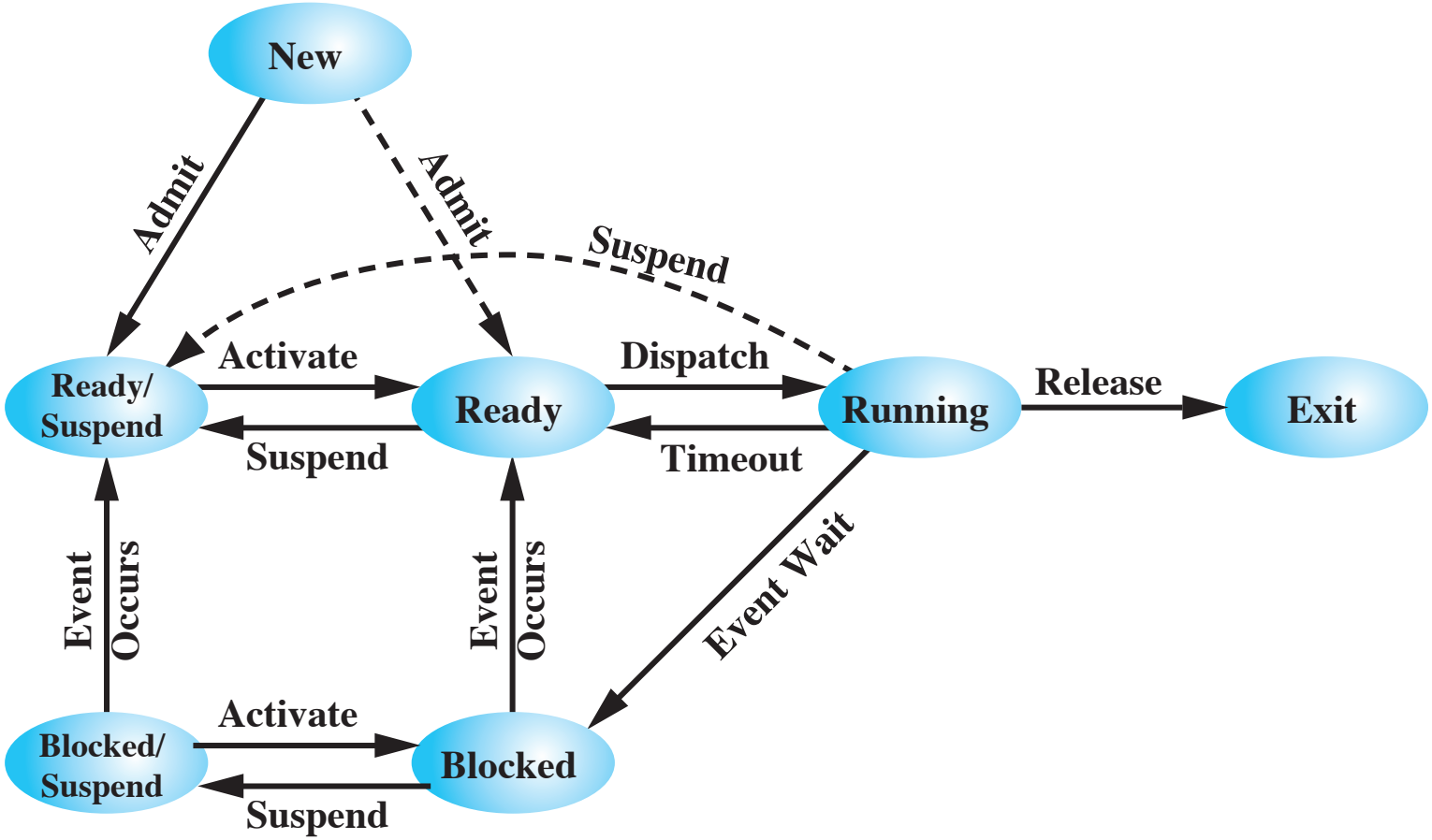


Figure 3.6 Five-State Process Model



(a) With One Suspend State

Figure 3.9 Process State Transition Diagram with Suspend States



(b) With Two Suspend States

Figure 3.9 Process State Transition Diagram with Suspend States

Modes of Execution

User Mode

- Less-privileged mode
- User programs typically execute in this mode

System Mode

- More-privileged mode
- Also referred to as control mode or kernel mode
- Kernel of the operating system