System Software Lab

Srividya Krishnakumar  
CS5A  
Roll No. 55

INDEX

1. Simulate the following non-preemptive CPU scheduling algorithms to find turn around time and waiting time: FCFS, SJF, Round Robin (Preemptive), Priority
2. Simulate working of a single level, two level and hierarchical directory structure.
3. Simulate Banker’s algorithm for deadlock detection.
4. Simulate the SCAN, C-SCAN and FCFS algorithms.
5. Implement the producer-consumer problem using semaphores.
6. Implement the Dining Philosopher’s problem
7. Implement pass 1 of a two-pass assembler.
8. Implement pass 2 of a two-pass assembler.
9. Implement a one-pass assembler.
10. Implement a two-pass macro processor.
11. Create a symbol table and use hashing to insert items.
12. Implement an absolute loader.
13. Simulate the following non-preemptive CPU scheduling algorithms to find turn around time and waiting time.

i) FCFS ii) SJF iii) Round Robin (Preemptive) iv)Priority

#include <stdio.h>

float calcAvg(int a[], int n)

{

int total = 0;

for (int i = 0; i < n; i++)

total += a[i];

return (float)total / n;

}

int find\_shortest\_job(int burst[], int arrival[], int n)

{

int min = 0;

int min\_burst = burst[0];

for (int i = 1; i < n; i++)

{

if (arrival[i] == 0 && burst[i] < min\_burst)

{

min = i;

min\_burst = burst[i];

}

}

return min;

}

void printDetails(int waiting[], int burst[], int n)

{

printf("Waiting times: ");

for (int i = 0; i < n; i++)

printf("%d ", waiting[i]);

printf("\nAverage waiting time is %.2f\n", calcAvg(waiting, n));

int turnaround[n];

printf("Turnaround times: ");

for (int i = 0; i < n; i++)

{

turnaround[i] = burst[i] + waiting[i];

printf("%d ", turnaround[i]);

}

printf("\nAverage turnaround time is %.2f\n", calcAvg(turnaround, n));

}

void cpu\_cycle(int burst[], int waiting[], int arrival[], int process\_in\_cpu, int n)

{

for (int i = 0; i < n; i++)

{

if (i != process\_in\_cpu && arrival[i] == 0)

waiting[i]++;

else if (i != process\_in\_cpu && arrival[i] > 0)

arrival[i]--;

}

burst[process\_in\_cpu]--;

}

void fcfs(int burst[], int arrival[], int n)

{

int burst\_backup[n];

for (int i = 0; i < n; i++)

burst\_backup[i] = burst[i];

int arrival\_backup[n];

for (int i = 0; i < n; i++)

arrival\_backup[i] = arrival[i];

int waiting[n];

for (int i = 0; i < n; i++)

waiting[i] = 0;

int arrival\_pointer = 0;

int process\_in\_cpu = -1;

while (1)

{

// Wait with CPU idling

while (arrival\_backup[arrival\_pointer] > 0 && process\_in\_cpu == -1)

{

arrival\_backup[arrival\_pointer]--;

continue;

}

process\_in\_cpu = arrival\_pointer;

while (burst\_backup[process\_in\_cpu] > 0)

cpu\_cycle(burst\_backup, waiting, arrival\_backup, process\_in\_cpu, n);

arrival\_backup[process\_in\_cpu] = -1;

process\_in\_cpu = -1;

arrival\_pointer++;

if (arrival\_pointer >= n)

break;

}

printDetails(waiting, burst, n);

}

void sjf(int burst[], int arrival[], int n)

{

int burst\_backup[n];

for (int i = 0; i < n; i++)

burst\_backup[i] = burst[i];

int arrival\_backup[n];

for (int i = 0; i < n; i++)

arrival\_backup[i] = arrival[i];

int waiting[n];

for (int i = 0; i < n; i++)

waiting[i] = 0;

int arrival\_pointer = 0;

int process\_in\_cpu = -1;

while (1)

{

// Wait with CPU idling

while (arrival\_backup[arrival\_pointer] > 0 && process\_in\_cpu == -1)

{

arrival\_backup[arrival\_pointer]--;

continue;

}

process\_in\_cpu = find\_shortest\_job(burst, arrival\_backup, n);

while (burst\_backup[process\_in\_cpu] > 0)

cpu\_cycle(burst\_backup, waiting, arrival\_backup, process\_in\_cpu, n);

arrival\_backup[process\_in\_cpu] = -1;

process\_in\_cpu = -1;

arrival\_pointer++;

if (arrival\_pointer >= n)

break;

}

printDetails(waiting, burst, n);

}

void round\_robin(int burst[], int arrival[], int n, int tq)

{

int burst\_backup[n];

for (int i = 0; i < n; i++)

burst\_backup[i] = burst[i];

int arrival\_backup[n];

for (int i = 0; i < n; i++)

arrival\_backup[i] = arrival[i];

int waiting[n];

for (int i = 0; i < n; i++)

waiting[i] = 0;

int arrival\_pointer = 0;

int process\_in\_cpu = -1;

while (1)

{

// Wait with CPU idling

while (arrival\_backup[arrival\_pointer] > 0 && process\_in\_cpu == -1)

{

/\* If no new processes have arrived and an existing process has burst time left,

execute that process instead of waiting for another process to arrive \*/

for (int i = 0; i < arrival\_pointer; i++)

if (burst\_backup[i] > 0)

{

arrival\_pointer = i;

break;

}

arrival\_backup[arrival\_pointer]--;

continue;

}

process\_in\_cpu = arrival\_pointer;

int cycles\_used = 0;

while (cycles\_used < tq && burst\_backup[process\_in\_cpu] > 0)

{

cpu\_cycle(burst\_backup, waiting, arrival\_backup, process\_in\_cpu, n);

cycles\_used++;

}

if (burst\_backup[process\_in\_cpu] == 0)

arrival\_backup[process\_in\_cpu] = -1;

process\_in\_cpu = -1;

arrival\_pointer++;

// Once all processes get a turn go back to first process

if (arrival\_pointer >= n)

arrival\_pointer = 0;

int all\_complete = 1;

// Stop only when burst for all processes is 0

for (int i = 0; i < n; i++)

{

if (burst\_backup[i] > 0)

all\_complete = 0;

}

if (all\_complete == 1)

break;

}

printDetails(waiting, burst, n);

}

void priority\_scheduling(int burst[], int arrival[], int n, int priorities[])

{

int burst\_backup[n];

for (int i = 0; i < n; i++)

burst\_backup[i] = burst[i];

int arrival\_backup[n];

for (int i = 0; i < n; i++)

arrival\_backup[i] = arrival[i];

int waiting[n];

for (int i = 0; i < n; i++)

waiting[i] = 0;

int arrival\_pointer = 0;

int process\_in\_cpu = -1;

while (1)

{

// Wait with CPU idling

while (arrival\_backup[arrival\_pointer] > 0 && process\_in\_cpu == -1)

{

arrival\_backup[arrival\_pointer]--;

continue;

}

int highest\_priority = 0;

for (int i = 0; i < n; i++)

if (arrival\_backup[i] == 0 && priorities[i] > highest\_priority)

{

highest\_priority = priorities[i];

process\_in\_cpu = i;

}

while (burst\_backup[process\_in\_cpu] > 0)

cpu\_cycle(burst\_backup, waiting, arrival\_backup, process\_in\_cpu, n);

arrival\_backup[process\_in\_cpu] = -1;

process\_in\_cpu = -1;

arrival\_pointer++;

if (arrival\_pointer >= n)

break;

}

printDetails(waiting, burst, n);

}

void main()

{

int n;

printf("Enter number of processes\n");

scanf("%d", &n);

int burst[n];

int arrival[n];

printf("Enter burst times of %d processes\n", n);

for (int i = 0; i < n; i++)

scanf("%d", &burst[i]);

printf("Enter arrival times of %d processes (should be in ascending order)\n", n);

for (int i = 0; i < n; i++)

scanf("%d", &arrival[i]);

printf("Enter time quantum (Only applicable to round robin scheduling\n");

int time\_quantum;

scanf("%d", &time\_quantum);

printf("Enter priorities of %d processes (Only applicable to priority scheduling)\n", n);

int priorities[n];

for (int i = 0; i < n; i++)

scanf("%d", &priorities[i]);

printf("\nUsing FCFS algorithm:\n");

fcfs(burst, arrival, n);

printf("\nUsing SJF algorithm\n");

sjf(burst, arrival, n);

printf("\nUsing Round Robin algorithm\n");

round\_robin(burst, arrival, n, time\_quantum);

printf("\nUsing priority scheduling\n");

priority\_scheduling(burst, arrival, n, priorities);

printf("\n");

}

OUTPUT:

Enter number of processes

3

Enter burst times of 3 processes

2 5 7

Enter arrival times of 3 processes (should be in ascending order)

0 2 4

Enter time quantum (Only applicable to round robin scheduling

2

Enter priorities of 3 processes (Only applicable to priority scheduling)

5 4 3

Using FCFS algorithm:

Waiting times: 0 0 3

Average waiting time is 1.00

Turnaround times: 2 5 10

Average turnaround time is 5.67

Using SJF algorithm

Waiting times: 0 0 0

Average waiting time is 0.00

Turnaround times: 2 5 7

Average turnaround time is 4.67

Using Round Robin algorithm

Average turnaround time is 7.00

Using priority scheduling

Waiting times: 0 0 3

Average waiting time is 1.00

Turnaround times: 2 5 10

Average turnaround time is 5.67

1. Simulate working of a single level, two level and hierarchical directory structure.

**Single level:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

struct files

{

char name[128];

struct files \*p;

} \* head, \*curr;

void printDirectory()

{

if (head == NULL)

{

printf("No files present!\n");

return;

}

struct files \*temp = head;

printf("\n");

while (temp)

{

printf("|\n");

printf("--%s\n", temp->name);

temp = temp->p;

}

}

void removeFile()

{

printf("Enter filename\n");

char fname[128];

scanf("%s", fname);

struct files \*temp = head;

if (strcmp(temp->name, fname) == 0)

{

head = temp->p;

printf("File deleted.\n");

return;

}

while (temp != NULL && temp->p != NULL)

{

if (strcmp(temp->p->name, fname) == 0)

{

temp->p = temp->p->p;

printf("File deleted.\n");

return;

}

temp = temp->p;

}

printf("File not found!\n");

}

void addFile()

{

printf("Enter filename\n");

char fname[128];

scanf("%s", fname);

if (curr == NULL)

{

curr = (struct files \*)malloc(sizeof(struct files));

strcpy(curr->name, fname);

curr->p = NULL;

head = curr;

return;

}

struct files \*temp = (struct files \*)malloc(sizeof(struct files));

strcpy(temp->name, fname);

temp->p = NULL;

curr->p = temp;

curr = temp;

}

void main()

{

int in;

while (true)

{

printf("\n\nYou are in the only directory present.\nEnter 1 to show directory\nEnter 2 to add new file\nEnter 3 to delete file\nEnter anything else to exit\n");

scanf("%d", &in);

switch (in)

{

case 1:

printDirectory();

break;

case 2:

addFile();

break;

case 3:

removeFile();

break;

default:

exit(0);

}

}

}

OUTPUT:

You are in the only directory present.

Enter 1 to show directory

Enter 2 to add new file

Enter 3 to delete file

Enter anything else to exit

2

Enter filename

hello.txt

You are in the only directory present.

Enter 1 to show directory

Enter 2 to add new file

Enter 3 to delete file

Enter anything else to exit

1

|

--hello.txt

You are in the only directory present.

Enter 1 to show directory

Enter 2 to add new file

Enter 3 to delete file

Enter anything else to exit

**Two-level directory:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

struct node

{

char name[128];

bool isDir;

struct node \*p;

struct node \*c[100];

int i;

int level;

} \* head, \*curr;

void ls()

{

if (curr->i == 0)

{

printf("Empty directory\n");

return;

}

for (int i = 0; i < curr->i; i++)

{

if (curr->c[i]->isDir)

printf("\*%s\* ", curr->c[i]->name);

else

printf("%s ", curr->c[i]->name);

}

}

void touch(bool d)

{

if (d && curr->level >= 1)

{

printf("Cannot create more than two levels of directories\n");

return;

}

if (d)

printf("Enter directory name\n");

else

printf("Enter filename\n");

char fname[128];

scanf("%s", fname);

struct node \*temp = (struct node \*)malloc(sizeof(struct node));

strcpy(temp->name, fname);

temp->isDir = d;

temp->p = curr;

temp->level = (curr->level) + 1;

curr->c[curr->i] = temp;

curr->i = (curr->i) + 1;

}

void cd()

{

printf("Enter directory name\n");

char dname[128];

scanf("%s", dname);

for (int i = 0; i < curr->i; i++)

{

if (!strcmp(curr->c[i]->name, dname) && curr->c[i]->isDir == true)

{

curr = curr->c[i];

return;

}

}

printf("Directory not present.\n");

}

void cdup()

{

if (curr->p == NULL)

{

printf("You are at the root directory\n");

return;

}

curr = curr->p;

}

void rm(bool d)

{

printf("Enter name of file or directory to delete\n");

char name[128];

scanf("%s", name);

for (int i = 0; i < curr->i; i++)

{

if (!strcmp(curr->c[i]->name, name) && ((d && curr->c[i]->isDir == true) || (!d && curr->c[i]->isDir == false)))

{

int t = i;

while (t < (curr->i) - 1)

{

curr->c[t] = curr->c[t + 1];

t++;

}

curr->i = (curr->i) - 1;

printf("Successfully deleted.\n");

return;

}

}

printf("Not found\n");

}

void main()

{

int in;

head = (struct node \*)malloc(sizeof(struct node));

strcpy(head->name, "root");

head->isDir = true;

head->p = NULL;

head->i = 0;

head->level = 0;

curr = head;

while (true)

{

printf("\n\nYou are in %s directory.\nEnter 1 to show everything in this directory\nEnter 2 to change directory\nEnter 3 to go to parent directory\nEnter 4 to add new file\nEnter 5 to delete file\nEnter 6 to create new directory\nEnter 7 to delete directory\nEnter 8 to exit\n", curr->name);

scanf("%d", &in);

switch (in)

{

case 1:

ls();

break;

case 2:

cd();

break;

case 3:

cdup();

break;

case 4:

touch(false);

break;

case 5:

rm(false);

break;

case 6:

touch(true);

break;

case 7:

rm(true);

break;

default:

exit(0);

}

}

}

OUTPUT:

You are in root directory.

Enter 1 to show everything in this directory

Enter 2 to change directory

Enter 3 to go to parent directory

Enter 4 to add new file

Enter 5 to delete file

Enter 6 to create new directory

Enter 7 to delete directory

Enter 8 to exit

6

Enter directory name

hello

You are in root directory.

Enter 1 to show everything in this directory

Enter 2 to change directory

Enter 3 to go to parent directory

Enter 4 to add new file

Enter 5 to delete file

Enter 6 to create new directory

Enter 7 to delete directory

Enter 8 to exit

2

Enter directory name

hello

You are in hello directory.

Enter 1 to show everything in this directory

Enter 2 to change directory

Enter 3 to go to parent directory

Enter 4 to add new file

Enter 5 to delete file

Enter 6 to create new directory

Enter 7 to delete directory

Enter 8 to exit

6

Cannot create more than two levels of directories

You are in hello directory.

Enter 1 to show everything in this directory

Enter 2 to change directory

Enter 3 to go to parent directory

Enter 4 to add new file

Enter 5 to delete file

Enter 6 to create new directory

Enter 7 to delete directory

Enter 8 to exit

**Hierarchical directory:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

struct node

{

char name[128];

bool isDir;

struct node \*p;

struct node \*c[100];

int i;

} \* head, \*curr;

void ls()

{

if (curr->i == 0)

{

printf("Empty directory\n");

return;

}

for (int i = 0; i < curr->i; i++)

{

if (curr->c[i]->isDir)

printf("\*%s\* ", curr->c[i]->name);

else

printf("%s ", curr->c[i]->name);

}

}

void touch(bool d)

{

if (d)

printf("Enter directory name\n");

else

printf("Enter filename\n");

char fname[128];

scanf("%s", fname);

struct node \*temp = (struct node \*)malloc(sizeof(struct node));

strcpy(temp->name, fname);

temp->isDir = d;

temp->p = curr;

curr->c[curr->i] = temp;

curr->i = (curr->i) + 1;

}

void cd()

{

printf("Enter directory name\n");

char dname[128];

scanf("%s", dname);

for (int i = 0; i < curr->i; i++)

{

if (!strcmp(curr->c[i]->name, dname) && curr->c[i]->isDir == true)

{

curr = curr->c[i];

return;

}

}

printf("Directory not present.\n");

}

void cdup()

{

if (curr->p == NULL)

{

printf("You are at the root directory\n");

return;

}

curr = curr->p;

}

void rm(bool d)

{

printf("Enter name of file or directory to delete\n");

char name[128];

scanf("%s", name);

for (int i = 0; i < curr->i; i++)

{

if (!strcmp(curr->c[i]->name, name) && ((d && curr->c[i]->isDir == true) || (!d && curr->c[i]->isDir == false)))

{

int t = i;

while (t < (curr->i) - 1)

{

curr->c[t] = curr->c[t + 1];

t++;

}

curr->i = (curr->i) - 1;

printf("Successfully deleted.\n");

return;

}

}

printf("Not found\n");

}

void main()

{

int in;

head = (struct node \*)malloc(sizeof(struct node));

strcpy(head->name, "root");

head->isDir = true;

head->p = NULL;

head->i = 0;

curr = head;

while (true)

{

printf("\n\nYou are in %s directory.\nEnter 1 to show everything in this directory\nEnter 2 to change directory\nEnter 3 to go to parent directory\nEnter 4 to add new file\nEnter 5 to delete file\nEnter 6 to create new directory\nEnter 7 to delete directory\nEnter 8 to exit\n", curr->name);

scanf("%d", &in);

switch (in)

{

case 1:

ls();

break;

case 2:

cd();

break;

case 3:

cdup();

break;

case 4:

touch(false);

break;

case 5:

rm(false);

break;

case 6:

touch(true);

break;

case 7:

rm(true);

break;

default:

exit(0);

}

}

}

OUTPUT:

You are in root directory.

Enter 1 to show everything in this directory

Enter 2 to change directory

Enter 3 to go to parent directory

Enter 4 to add new file

Enter 5 to delete file

Enter 6 to create new directory

Enter 7 to delete directory

Enter 8 to exit

6

Enter directory name

hello

You are in root directory.

Enter 1 to show everything in this directory

Enter 2 to change directory

Enter 3 to go to parent directory

Enter 4 to add new file

Enter 5 to delete file

Enter 6 to create new directory

Enter 7 to delete directory

Enter 8 to exit

2

Enter directory name

hello

You are in hello directory.

Enter 1 to show everything in this directory

Enter 2 to change directory

Enter 3 to go to parent directory

Enter 4 to add new file

Enter 5 to delete file

Enter 6 to create new directory

Enter 7 to delete directory

Enter 8 to exit

3

You are in root directory.

Enter 1 to show everything in this directory

Enter 2 to change directory

Enter 3 to go to parent directory

Enter 4 to add new file

Enter 5 to delete file

Enter 6 to create new directory

Enter 7 to delete directory

Enter 8 to exit

4

Enter filename

test

You are in root directory.

Enter 1 to show everything in this directory

Enter 2 to change directory

Enter 3 to go to parent directory

Enter 4 to add new file

Enter 5 to delete file

Enter 6 to create new directory

Enter 7 to delete directory

Enter 8 to exit

1

\*hello\* test

You are in root directory.

Enter 1 to show everything in this directory

Enter 2 to change directory

Enter 3 to go to parent directory

Enter 4 to add new file

Enter 5 to delete file

Enter 6 to create new directory

Enter 7 to delete directory

Enter 8 to exit

1. Simulate Banker’s algorithm for deadlock detection.

#include <stdio.h>

#include <stdbool.h>

bool check\_if\_resources\_are\_enough(int \*res, int res\_left[], int i, int n, int r)

{

for (int k = 0; k < r; k++)

if (\*(res + i \* r + k) > res\_left[k])

return false;

return true;

}

void bankers\_algo(int \*res\_needed, int \*res\_allocated, int res\_left[], int n, int r, int finished\_processes)

{

bool deadlock\_present;

while (true)

{

deadlock\_present = true;

for (int i = 0; i < n; i++)

{

if (\*(res\_needed + i \* r) != -1 && check\_if\_resources\_are\_enough(res\_needed, res\_left, i, n, r))

{

for (int j = 0; j < r; j++)

res\_left[j] = \*(res\_needed + i \* r + j) + \*(res\_allocated + i \* r + j);

finished\_processes++;

if (finished\_processes == n)

{

printf("Deadlock is not present\n");

return;

}

\*(res\_needed + i \* r) = -1;

deadlock\_present = false;

}

}

if (deadlock\_present)

{

printf("Deadlock Present!\n");

return;

}

}

}

void main()

{

printf("\*\* Program to simulate Banker's algorithm \*\*\n");

int n;

printf("Enter number of processes\n");

scanf("%d", &n);

int r;

printf("Enter number of resources\n");

scanf("%d", &r);

int res\_needed[n][r];

int res\_allocated[n][r];

int res\_left[r];

printf("Enter resources currently NEEDED by %d processes\n", n);

for (int i = 0; i < n; i++)

for (int j = 0; j < r; j++)

scanf("%d", (\*(res\_needed + i) + j));

printf("Enter resources currently HELD by %d processes\n", n);

for (int i = 0; i < n; i++)

for (int j = 0; j < r; j++)

scanf("%d", (\*(res\_allocated + i) + j));

printf("Enter amount of resources that are left\n");

for (int i = 0; i < r; i++)

scanf("%d", res\_left + i);

int finished\_processes = 0;

bankers\_algo(res\_needed[0], res\_allocated[0], res\_left, n, r, finished\_processes);

}

OUTPUT:

\*\* Program to simulate Banker's algorithm \*\*

Enter number of processes

3

Enter number of resources

2

Enter resources currently NEEDED by 3 processes

2 3

0 1

1 1

Enter resources currently HELD by 3 processes

0 0

0 0

0 0

Enter amount of resources that are left

1 2

Deadlock Present!

1. Simulate the SCAN, C-SCAN and FCFS algorithms.

#include <stdio.h>

#include <stdlib.h>

int cmpfunc(const void \*a, const void \*b)

{

return (\*(int \*)a - \*(int \*)b);

}

void fcfs(int \*locs, int n, int start)

{

int seek\_distance = 0;

seek\_distance += abs(start - locs[0]);

for (int i = 1; i < n; i++)

seek\_distance += abs(locs[i] - locs[i - 1]);

printf("Average movement of head using FCFS: %.3f\n", (float)seek\_distance / n);

}

void scan(int \*locs, int n, int start, int max)

{

int seek\_distance = 0;

int \*temp = locs;

qsort(temp, n, sizeof(int), cmpfunc);

if (temp[0] > start)

printf("Average movement of head using SCAN: %.3f\n", (float)(abs(temp[n - 1] - start) / n));

else

printf("Average movement of head using SCAN: %.3f\n", (float)((max - temp[0] + max - start) / n));

}

void cscan(int \*locs, int n, int start, int max)

{

int seek\_distance = 0;

int \*temp = locs;

qsort(temp, n, sizeof(int), cmpfunc);

if (temp[0] > start)

printf("Average movement of head using C-SCAN: %.3f\n", (float)(abs(temp[n - 1] - start) / n));

else

{

int i;

for (i = 0; i < n; i++)

if (locs[i] > start)

break;

i -= 1;

printf("Average movement of head using C-SCAN: %.3f\n", (float)((max - start + max + locs[i]) / n));

}

}

void main()

{

printf("Enter number of locations\n");

int n;

scanf("%d", &n);

printf("Enter starting location of head\n");

int start;

scanf("%d", &start);

printf("Enter maximum possible location index\n");

int max;

scanf("%d", &max);

printf("Enter the %d locations on the disk to access data from\n", n);

int locs[n];

for (int i = 0; i < n; i++)

{

scanf("%d", locs + i);

if (\*(locs + i) > max)

{

printf("ERROR: Location greater than maximum location possible\n");

return;

}

}

fcfs(locs, n, start);

scan(locs, n, start, max);

cscan(locs, n, start, max);

}

OUTPUT:

Enter number of locations

3

Enter starting location of head

40

Enter maximum possible location index

100

Enter the 3 locations on the disk to access data from

50

20

90

Average movement of head using FCFS: 36.667

Average movement of head using SCAN: 46.000

Average movement of head using C-SCAN: 60.000

1. Implement the producer-consumer problem using semaphores

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()

{

int n;

printf("Enter the max size of the buffer\n");

scanf("%d", &n);

int buffer[n];

int i = -1;

int c;

do

{

printf("\nChoose:\n1.Produce\n2.Consume\n3.Exit\n");

scanf("%d", &c);

switch (c)

{

case 1:

if (i < n - 1)

{

int data;

printf("Enter data to produce\n");

scanf("%d", &data);

i++;

buffer[i] = data;

}

else

printf("Semaphore is full!\n");

break;

case 2:

if (i >= 0)

{

printf("Data consumed is %d\n", buffer[i]);

i--;

}

else

printf("Semaphore is empty!\n");

break;

default:

exit(0);

}

} while (c < 3);

}

OUTPUT:

Enter the max size of the buffer

2

Choose:

1.Produce

2.Consume

3.Exit

1

Enter data to produce

12

Choose:

1.Produce

2.Consume

3.Exit

1

Enter data to produce

15

Choose:

1.Produce

2.Consume

3.Exit

1

Semaphore is full!

Choose:

1.Produce

2.Consume

3.Exit

2

Data consumed is 15

Choose:

1.Produce

2.Consume

3.Exit

2

Data consumed is 12

Choose:

1.Produce

2.Consume

3.Exit

2

Semaphore is empty!

1. Implement the Dining Philosopher’s problem

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

bool all\_philosophers\_finished\_eating(bool phils[], int n)

{

for (int i = 0; i < n; i++)

if (!phils[i])

return false;

return true;

}

void clear\_chopsticks(bool chops[], int n)

{

for (int i = 0; i < n; i++)

chops[i] = true;

}

int main()

{

printf("Enter number of philosophers\n");

int n;

scanf("%d", &n);

bool chops[n];

bool philosophers\_finished\_eating[n];

clear\_chopsticks(chops, n);

bool flag = true;

while (flag)

{

printf("\nNew loop:\n");

clear\_chopsticks(chops, n);

flag = false;

for (int i = 0; i < n; i++)

{

if (!philosophers\_finished\_eating[i])

{

if (chops[i] && chops[(i + 1) % 5])

{

chops[i] = false;

chops[(i + 1) % 5] = false;

printf("Philosopher %d is eating\n", i);

philosophers\_finished\_eating[i] = true;

flag = true;

}

else

printf("Philosopher %d is thinking\n", i);

}

else

printf("Philosopher %d has finished eating\n", i);

}

if (all\_philosophers\_finished\_eating(philosophers\_finished\_eating, n))

{

printf("Program completed successfully\n");

exit(0);

}

}

printf("Deadlock is present\n");

}

OUTPUT:

Enter number of philosophers

5

New loop:

Philosopher 0 is eating

Philosopher 1 is thinking

Philosopher 2 is eating

Philosopher 3 is thinking

Philosopher 4 is thinking

New loop:

Philosopher 0 has finished eating

Philosopher 1 is eating

Philosopher 2 has finished eating

Philosopher 3 is eating

Philosopher 4 is thinking

New loop:

Philosopher 0 has finished eating

Philosopher 1 has finished eating

Philosopher 2 has finished eating

Philosopher 3 has finished eating

Philosopher 4 is eating

Program completed successfully

1. Implement pass 1 of a two-pass assembler.

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void main()

{

FILE \*inp,\*optab,\*symtab,\*f4;

int locctr,starting\_addr,l,operand,o,len;

char opcode[20],label[20],op[20],opcode\_from\_optable[20];

inp=fopen("inp.txt","r");

symtab=fopen("symtab.txt","w");

fscanf(inp,"%s %s %d",label,opcode,&operand);

if(strcmp(opcode,"START")==0)

{

starting\_addr=operand;

locctr=starting\_addr;

printf("\t%s\t%s\t%d\n",label,opcode,operand);

}

else

locctr=0;

fscanf(inp,"%s %s",label,opcode);

while(!feof(inp))

{

fscanf(inp,"%s",op);

printf("\n%d\t%s\t%s\t%s\n",locctr,label,opcode,op);

if(strcmp(label,"-")!=0)

{

fprintf(symtab,"\n%d\t%s\t%s\t%s\n",locctr,label,opcode,op);

}

optab=fopen("optab.txt","r");

fscanf(optab,"%s %d",opcode\_from\_optable,&o);

while(!feof(optab))

{

if(strcmp(opcode,opcode\_from\_optable)==0)

{

locctr=locctr+3;

break;

}

fscanf(optab,"%s %d",opcode\_from\_optable,&o);

}

fclose(optab);

if(strcmp(opcode,"WORD")==0)

{

locctr=locctr+3;

}

else if(strcmp(opcode,"RESW")==0)

{

operand=atoi(op);

locctr=locctr+(3\*operand);

}

else if(strcmp(opcode,"BYTE")==0)

{

if(op[0]=='X')

locctr=locctr+1;

else

{

len=strlen(op)-3;

locctr=locctr+len;

}

}

else if(strcmp(opcode,"RESB")==0)

{

operand=atoi(op);

locctr=locctr+operand;

}

fscanf(inp,"%s%s",label,opcode);

}

if(strcmp(opcode,"END")==0)

{

printf("\nProgram Length = %d",locctr-starting\_addr);

}

fclose(inp);

fclose(symtab);

}

inp.txt:

COPY START 1000

- LDA ALPHA

- ADD ONE

- SUB TWO

- STA BETA

ALPHA BYTE C'AGZ'

ONE RESB 2

TWO WORD 5

BETA RESW 1

- END -

optab.txt:

LDA 00

STA 23

ADD 01

SUB 05

symtab.txt:

1009 ALPHA BYTE C'AGZ'

1012 ONE RESB 2

1014 TWO WORD 5

1017 BETA RESW 1

1. Implement pass 2 of a two-pass assembler.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void main()

{

FILE \*fint, \*ftab, \*flen, \*fsym;

int op1[10], txtlen, txtlen1, i, j = 0, len;

char add[5], symadd[5], op[5], start[10], temp[30], line[20], label[20], mne[10], operand[10], symtab[10], opmne[10];

fint = fopen("input.txt", "r");

flen = fopen("length.txt", "r");

ftab = fopen("optab.txt", "r");

fsym = fopen("symbol.txt", "r");

fscanf(fint, "%s%s%s%s", add, label, mne, operand);

if (strcmp(mne, "START") == 0)

{

strcpy(start, operand);

fscanf(flen, "%d", &len);

}

printf("H^%s^%s^%d\nT^00%s^", label, start, len, start);

fscanf(fint, "%s%s%s%s", add, label, mne, operand);

while (strcmp(mne, "END") != 0)

{

fscanf(ftab, "%s%s", opmne, op);

while (!feof(ftab))

{

if (strcmp(mne, opmne) == 0)

{

fclose(ftab);

fscanf(fsym, "%s%s", symadd, symtab);

while (!feof(fsym))

{

if (strcmp(operand, symtab) == 0)

{

printf("%s%s^", op, symadd);

break;

}

else

fscanf(fsym, "%s%s", symadd, symtab);

}

break;

}

else

fscanf(ftab, "%s%s", opmne, op);

}

if ((strcmp(mne, "BYTE") == 0) || (strcmp(mne, "WORD") == 0))

{

if (strcmp(mne, "WORD") == 0)

printf("0000%s^", operand);

else

{

len = strlen(operand);

for (i = 2; i < len; i++)

{

printf("%d", operand[i]);

}

printf("^");

}

}

fscanf(fint, "%s%s%s%s", add, label, mne, operand);

ftab = fopen("optab.txt", "r");

fseek(ftab, SEEK\_SET, 0);

}

printf("\nE^00%s", start);

fclose(fint);

fclose(ftab);

fclose(fsym);

fclose(flen);

}

OUTPUT:

H^COPY^1000^25

T^001000^001012^011017^7576786769^00005^

E^001000

1. Implement a one-pass assembler.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void main()

{

FILE \*f1, \*f2, \*f3, \*f4, \*f5;

int lc, sa, i = 0, j = 0, m[10], pgmlen, len, k, len1, l = 0;

char name[10], opnd[10], la[10], mne[10], s1[10], mne1[10], opnd1[10];

char lcs[10], ms[10];

char sym[10], symaddr[10], obj1[10], obj2[10], s2[10], q[10], s3[10];

f1 = fopen("input.txt", "r");

f2 = fopen("optab.txt", "r");

f3 = fopen("symtab.txt", "w+");

f4 = fopen("symtab1.txt", "w+");

f5 = fopen("output.txt", "w+");

fscanf(f1, "%s%s%s", la, mne, opnd);

if (strcmp(mne, "START") == 0)

{

sa = atoi(opnd);

strcpy(name, la);

lc = sa;

}

strcpy(s1, "\*");

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "END") != 0)

{

if (strcmp(la, "-") == 0)

{

fscanf(f2, "%s%s", mne1, opnd1);

while (!feof(f2))

{

if (strcmp(mne1, mne) == 0)

{

m[i] = lc + 1;

fprintf(f3, "%s\t%s\n", opnd, s1);

fprintf(f5, "%s\t0000\n", opnd1);

lc = lc + 3;

i = i + 1;

break;

}

else

fscanf(f2, "%s%s", mne1, opnd1);

}

}

else

{

fseek(f3, SEEK\_SET, 0);

fscanf(f3, "%s%s", sym, symaddr);

while (!feof(f3))

{

if (strcmp(sym, la) == 0)

{

sprintf(lcs, "%d", lc);

fprintf(f4, "%s\t%s\n", la, lcs);

sprintf(ms, "%d", m[j]);

j = j + 1;

fprintf(f5, "%s\t%s\n", ms, lcs);

i = i + 1;

break;

}

else

fscanf(f3, "%s%s", sym, symaddr);

}

if (strcmp(mne, "RESW") == 0)

lc = lc + 3 \* atoi(opnd);

else if (strcmp(mne, "BYTE") == 0)

{

strcpy(s2, "-");

len = strlen(opnd);

lc = lc + len - 2;

for (k = 2; k < len; k++)

{

q[l] = opnd[k];

l = l + 1;

}

fprintf(f5, "%s\t%s\n", q, s2);

break;

}

else if (strcmp(mne, "RESB") == 0)

lc = lc + atoi(opnd);

else if (strcmp(mne, "WORD") == 0)

{

strcpy(s3, "#");

lc = lc + 3;

fprintf(f5, "%s\t%s\n", opnd, s3);

break;

}

}

fseek(f2, SEEK\_SET, 0);

fscanf(f1, "%s%s%s", la, mne, opnd);

}

fseek(f5, SEEK\_SET, 0);

pgmlen = lc - sa;

printf("H^%s^%d^0%x\n", name, sa, pgmlen);

printf("T^");

printf("00%d^0%x", sa, pgmlen);

fscanf(f5, "%s%s", obj1, obj2);

while (!feof(f5))

{

if (strcmp(obj2, "0000") == 0)

printf("^%s%s", obj1, obj2);

else if (strcmp(obj2, "-") == 0)

{

printf("^");

len1 = strlen(obj1);

for (k = 0; k < len1; k++)

printf("%d", obj1[k]);

}

else if (strcmp(obj2, "#") == 0)

{

printf("^");

printf("%s", obj1);

}

fscanf(f5, "%s%s", obj1, obj2);

}

fseek(f5, SEEK\_SET, 0);

fscanf(f5, "%s%s", obj1, obj2);

while (!feof(f5))

{

if (strcmp(obj2, "0000") != 0)

{

if (strcmp(obj2, "-") != 0)

{

if (strcmp(obj2, "#") != 0)

{

printf("\n");

printf("T^%s^02^%s", obj1, obj2);

}

}

}

fscanf(f5, "%s%s", obj1, obj2);

}

printf("\nE^00%d\n", sa);

}

input.txt:

COPY START 1000

- LDA ALPHA

- STA BETA

ALPHA RESW 1

BETA RESW 1

- END -

optab.txt:

LDA 00

STA 23

LDCH 15

STCH 18

symtab.txt:

ALPHA \*

BETA \*

symtab1.txt:

ALPHA 1006

BETA 1009

output.txt:

00 0000

23 0000

1001 1006

1004 1009

1. Implement a two-pass macro processor.

Pass 1:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void main()

{

FILE \*f1, \*f2, \*f3;

char mne[20], opnd[20], la[20];

f1 = fopen("inp.txt", "r");

f2 = fopen("namtab.txt", "w+");

f3 = fopen("argtab.txt", "w+");

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "MEND") != 0)

{

if (strcmp(mne, "MACRO") == 0)

{

fprintf(f2, "%s\n", la);

fprintf(f3, "%s\t%s\n", la, opnd);

}

else

fprintf(f3, "%s\t%s\n", mne, opnd);

fscanf(f1, "%s%s%s", la, mne, opnd);

}

fprintf(f3, "%s", mne);

fclose(f1);

fclose(f2);

fclose(f3);

printf("Pass 1 is completed\n");

}

inp.txt:

EX1 MACRO &A,&B

- LDA &A

- STA &B

- MEND -

SAMPLE START 1000

- EX1 N1,N2

N1 RESW 1

N2 RESW 1

- END -

namtab.txt:

EX1

argtab.txt:

EX1 &A,&B

LDA &A

STA &B

MEND

Pass 2:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void main()

{

FILE \*f1, \*f2, \*f3, \*f4, \*f5;

int i, len;

char mne[20], opnd[20], la[20], name[20], mne1[20], opnd1[20], arg[20];

f1 = fopen("inp.txt", "r");

f2 = fopen("namtab.txt", "r");

f3 = fopen("argtab.txt", "r");

f4 = fopen("atab2.txt", "w+");

f5 = fopen("op2.txt", "w");

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "END") != 0)

{

if (strcmp(mne, "MACRO") == 0)

{

fscanf(f1, "%s%s%s", la, mne, opnd);

while (strcmp(mne, "MEND") != 0)

fscanf(f1, "%s%s%s", la, mne, opnd);

}

else

{

fscanf(f2, "%s", name);

if (strcmp(mne, name) == 0)

{

len = strlen(opnd);

for (i = 0; i < len; i++)

{

if (opnd[i] != ',')

fprintf(f4, "%c", opnd[i]);

else

fprintf(f4, "\n");

}

fseek(f2, SEEK\_SET, 0);

fseek(f4, SEEK\_SET, 0);

fscanf(f3, "%s%s", mne1, opnd1);

fprintf(f5, ".\t%s\t%s\n", mne1, opnd);

fscanf(f3, "%s%s", mne1, opnd1);

while (strcmp(mne1, "MEND") != 0)

{

if ((opnd1[0] == '&'))

{

fscanf(f4, "%s", arg);

fprintf(f5, "-\t%s\t%s\n", mne1, arg);

}

else

fprintf(f5, "-\t%s\t%s\n", mne1, opnd1);

fscanf(f3, "%s%s", mne1, opnd1);

}

}

else

fprintf(f5, "%s\t%s\t%s\n", la, mne, opnd);

}

fscanf(f1, "%s%s%s", la, mne, opnd);

}

fprintf(f5, "%s\t%s\t%s\n", la, mne, opnd);

fclose(f1);

fclose(f2);

fclose(f3);

fclose(f4);

fclose(f5);

printf("pass2");

}

atab2.txt:

N1

N2

op2.txt:

SAMPLE START 1000

- EX1 N1,N2

- LDA N1

- STA N2

N1 RESW 1

N2 RESW 1

- END -

1. Create a symbol table and use hashing to insert items.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define LENGTH 7

struct hashTable

{

char label[10];

int addr;

} ht[LENGTH];

void addLabel()

{

int addr;

char label[10];

printf("Enter label name\n");

scanf("%s", label);

printf("Enter label address\n");

scanf("%d", &addr);

int loc = addr % LENGTH;

if (ht[loc].addr == -1)

{

ht[loc].addr = addr;

strcpy(ht[loc].label, label);

}

else

printf("Hashtable slot occupied\n");

}

void display()

{

for (int i = 0; i < LENGTH; i++)

if (ht[i].addr != -1)

printf("%d %s\n", ht[i].addr, ht[i].label);

else

printf("0 0\n");

}

void main()

{

for (int i = 0; i < LENGTH; i++)

{

ht[i].addr = -1;

strcpy(ht[i].label, "");

}

int c = 0;

while (c < 3)

{

printf("Enter 1 to add label\nEnter 2 to view hashtable\n");

scanf("%d", &c);

switch (c)

{

case 1:

addLabel();

break;

case 2:

display();

}

}

}

OUTPUT:

Enter 1 to add label

Enter 2 to view hashtable

1

Enter label name

ALPHA

Enter label address

1000

Enter 1 to add label

Enter 2 to view hashtable

1

Enter label name

BETA

Enter label address

1003

Enter 1 to add label

Enter 2 to view hashtable

2

0 0

0 0

1003 BETA

0 0

0 0

0 0

1000 ALPHA

1. Implement an absolute loader.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void main()

{

FILE \*fp;

int addr, staddri;

char line[50], staddr[10];

fp = fopen("object\_code.txt", "r");

fscanf(fp, "%s", line);

while (!feof(fp))

{

fscanf(fp, "%s", line);

if (line[0] == 'T')

{

int i = 0, j = 0;

for (i = 2, j = 0; i < 8; i++, j++)

staddr[j] = line[i];

staddr[j] = '\0';

staddri = atoi(staddr);

i = 12;

while (line[i] != '$')

{

if (line[i] != '^')

{

printf("00%d %c%c\n", staddri, line[i], line[i + 1]);

staddri++;

i += 2;

}

else

i++;

}

}

else if (line[0] == 'E')

break;

}

}

object\_code.txt:

H^SAMPLE^001000^0035

T^001000^0C^001003^071009$

T^002000^03^111111$

E^001000

OUTPUT:

001000 00

001001 10

001002 03

001003 07

001004 10

001005 09

002000 11

002001 11

002002 11