slip 1	ſ	1	int pid;
1->	ι int i;	sp = min_i;	while (1)
#include <stdio.h></stdio.h>	for (i = 0; i < n; i++)	frames[sp] =	{
#define MAX 20	{	ref[i];	printf("myshell\$
int frames[MAX],	if (frames[i] ==	time[sp] = i;	");
ref[MAX],	pno)	count[sp] = 1;	fflush(stdin);
mem[MAX][MAX],	return i;	faults++;	fgets(buff, 80,
faults, sp, m, n,	}	for $(j = 0; j < n;$	stdin);
time[MAX];	return -1;	j++)	buff[strlen(buff) -
void accept()	}	mem[j][i] =	1] = '\0';
{	void lfu()	frames[j];	make_toks(buff,
int i;	{	}	args);
printf("Enter the	int i, j, k, count[20];	else	if (strcmp(args[0],
number of frames: ");	for (i = 0; i < m &&	{	"typeline") == 0)
scanf("%d", &n);	sp < n; i++)	time[k] = i;	{
printf("Enter the	{	count[k]++;	if (args[1][0] ==
number of references:	k = search(ref[i]);	}	'+')
");	if (k == -1)	}	{
scanf("%d", &m);	{	}	int n =
printf("Enter the	frames[sp] =	int main()	atoi(args[1] + 1);
reference string:\n");	ref[i];	{	atol(a185[±] · ±/)
for (i = 0; i < m; i++)	time[sp] = 1;	accept();	FILE *fp =
{	count[sp] = 1;	Ifu();	fopen(args[2], "r");
printf("[%d] = ", i);	faults++;	disp();	if (fp == NULL)
scanf("%d",	sp++;	return 0;	{
&ref[i]);	for (j = 0; j < n;	}	· ·
}	j++)	j	printf("Error opening
}	mem[j][i] =	2->	file %s.\n", args[2]);
void disp()	frames[j];	#include <sys types.h=""></sys>	return 1;
{	}	#include <sys stat.h=""></sys>	}
int i, j;	else	#include <fcntl.h></fcntl.h>	,
for (i = 0; i < m; i++)	{	#include <stdio.h></stdio.h>	char line[80];
printf("%3d",	time[k] = i;	#include <stdlib.h></stdlib.h>	int count = 0;
ref[i]);	count[k]++;	#include <unistd.h></unistd.h>	while (count <
printf("\n\n");	}	#include <string.h></string.h>	n && fgets(line, 80, fp)
for (i = 0; i < n; i++)	}	void make_toks(char	!= NULL)
{	, for (; i < m; i++)	*s, char *tok[])	{
for (j = 0; j < m;	{	{	printf("%s",
j++)	k = search(ref[i]);	int i = 0;	line);
, , {	if (k == -1)	char *p;	count++;
if (mem[i][j])	{	p = strtok(s, " ");	}
printf("%3d",	int min_i = 0,	while (p != NULL)	,
mem[i][j]);	min = 9999;	{	fclose(fp);
else	for (j = 0; j < n;	tok[i++] = p;	}
printf(" ");	j++)	p = strtok(NULL, "	,
j (<i>//</i>	, {	");	else if
printf("\n");	if (count[j] <	}	(strcmp(args[2], "-a")
}	min)	tok[i] = NULL;	== 0)
printf("Total Page	, {	}	-, {
Faults: %d\n", faults);	min =	int main()	·
}	count[j];	{	FILE *fp =
}	min_i = j;	char buff[80],	fopen(args[2], "r");
int search(int pno)	}	*args[10];	if (fp == NULL)
	,	O-11/	(. [- 1.022]

{	int i, j;	{	printf("%s\n",
·	for (i = 0; i < m; i++)	accept();	entry->d_name);
printf("Error opening	printf("%3d",	fifo();	break;
file %s.\n", args[2]);	ref[i]);	disp();	case 'n':
return 1;	printf("\n\n");	return 0;	count++;
}	for (i = 0; i < n; i++)	}	break;
j	{	J	}
char line[80];	for (j = 0; j < m;	2->	, l
while	j++)	#include <unistd.h></unistd.h>	J
(fgets(line, 80, fp) !=	, , , , , , , , , , , , , , , , , , ,	#include <sys types.h=""></sys>	closedir(dir);
NULL)	۱ if (mem[i][j])	#include <sys types.ii=""> #include <sys stat.h=""></sys></sys>	if (op == 'n')
printf("%s",	printf("%3d",	#include <fcntl.h></fcntl.h>	1 (OP == 11)
line);	mem[i][j]);	#include <stdio.h></stdio.h>	ι printf("Number of
iiie),	else	#include <stdlib.h></stdlib.h>	entries: %d\n", count);
foloso(fp):	printf(" ");	#include <string.h></string.h>	i entries. //attr / county,
fclose(fp);	βιπα(<i>),</i>	#include <string.n> #include <dirent.h></dirent.h></string.n>) 1
else	} printf("\p"\.	void make_toks(char	int main()
eise	printf("\n");	- ·	int main()
{ 	}	*s, char *tok[])	(
printf("Invalid	printf("Total Page	{ :=+:=0:	char buff[80],
option.\n");	Faults: %d\n", faults);	int i = 0;	*args[10];
}	}	char *p;	int pid;
}	int search(int pno)	p = strtok(s, " ");	while (1)
}	{	while (p != NULL)	{
}	int i;	{	printf("myshell\$
	for (i = 0; i < n; i++)	tok[i++] = p;	");
slip 2	{	p = strtok(NULL, "	fflush(stdin);
1->	if (frames[i] ==	");	fgets(buff, 80,
#include <stdio.h></stdio.h>	pno)	}	stdin);
#define MAX 20	return i;	tok[i] = NULL;	buff[strlen(buff) -
int frames[MAX],	}	}	1] = '\0';
ref[MAX],	return -1;	void list(char	make_toks(buff,
mem[MAX][MAX],	}	*dirname, char op)	args);
faults, sp, m, n;	void fifo()	{	if (strcmp(args[0],
void accept()	{	DIR *dir;	"list") == 0)
{	int i, j;	struct dirent *entry;	{
int i;	for (i = 0; i < m; i++)	int count = 0;	list(args[2],
printf("Enter no.of	{		args[1][0]);
frames:");	if (search(ref[i]) ==	dir =	}
scanf("%d", &n);	-1)	opendir(dirname);	else
printf("Enter no.of	{	if (dir == NULL)	{
references:");	frames[sp] =	{	pid = fork();
scanf("%d", &m);	ref[i];	printf("Directory	if (pid > 0)
printf("Enter	sp = (sp + 1) %	%s not found.\n",	wait();
reference string:\n");	n;	dirname);	else
for (i = 0; i < m; i++)	faults++;	return;	{
{	for $(j = 0; j < n;$	}	if
printf("[%d]=", i);	j++)	while ((entry =	(execvp(args[0], args)
scanf("%d",	mem[j][i] =	readdir(dir)) != NULL)	== -1)
&ref[i]);	frames[j];	{	printf("Bad
}	}	switch (op)	command.\n");
}	}	{	}
void disp()	}	case 'f':	}
{	int main()		}

return 0;	int search(int pno)	frames[sp] =	printf("File %s not
}	{	ref[i];	found.\n", fn);
	int i;	time[sp] = i;	return;
slip 3	for (i = 0; i < n; i++)	faults++;	}
1->	{	for (j = 0; j < n;	while ((c = fgetc(fh))
#include <stdio.h></stdio.h>	if (frames[i] ==	j++)	!= EOF)
#define MAX 20	pno)	mem[j][i] =	{
int frames[MAX],	return i;	frames[j];	if (c == ' ')
ref[MAX],	}	}	wc++;
mem[MAX][MAX],	return -1;	else	else if (c == '\n')
faults,	}	time[k] = i;	{
sp, m, n, time[MAX];	int get_Iru()	}	wc++;
void accept()	S = {	}	lc++;
{	int i, min_i, min =	int main()	}
int i;	9999;	{	, CC++;
printf("Enter no.of	for (i = 0; i < n; i++)	accept();	}
frames:");	{	Iru();	fclose(fh);
scanf("%d", &n);	if (time[i] < min)	disp();	switch (op)
printf("Enter no.of	{	return 0;	{
references:");	min = time[i];	}	case 'c':
scanf("%d", &m);	min_i = i;	•	printf("No.of
printf("Enter	}	2->	characters:%d\n", cc -
reference string:\n");	}	#include <sys types.h=""></sys>	1);
for (i = 0; i < m; i++)	return min_i;	#include <sys stat.h=""></sys>	break;
{	}	#include <fcntl.h></fcntl.h>	case 'w':
printf("[%d]=", i);	void Iru()	#include <stdio.h></stdio.h>	printf("No.of
scanf("%d",	{	#include <stdlib.h></stdlib.h>	words:%d\n", wc);
&ref[i]);	int i, j, k;	#include <unistd.h></unistd.h>	break;
}	for (i = 0; i < m &&	#include <string.h></string.h>	case 'l':
}	sp < n; i++)	void make_toks(char	printf("No.of
void disp()	{	*s, char *tok[])	lines:%d\n", lc + 1);
{	<pre>k = search(ref[i]);</pre>	{	break;
int i, j;	if (k == -1)	int i = 0;	}
for (i = 0; i < m; i++)	{	char *p;	}
printf("%3d",	frames[sp] =	p = strtok(s, " ");	int main()
ref[i]);	ref[i];	while (p != NULL)	{
printf("\n\n");	time[sp] = i;	{	char buff[80],
for (i = 0; i < n; i++)	faults++;	tok[i++] = p;	*args[10];
{	sp++;	p = strtok(NULL, "	int pid;
for $(j = 0; j < m;$	for $(j = 0; j < n;$	");	while (1)
j++)	j++)	}	{
{	mem[j][i] =	tok[i] = NULL;	printf("myshell\$
if (mem[i][j])	frames[j];	}	");
printf("%3d",	}	void count(char *fn,	fflush(stdin);
mem[i][j]);	else	char op)	fgets(buff, 80,
else	time[k] = i;	{	stdin);
printf(" ");	}	FILE *fh;	buff[strlen(buff) -
}	for (; i < m; i++)	int $cc = 0$, $wc = 0$, $lc =$	1] = '\0';
printf("\n");	{	0;	make_toks(buff,
}	<pre>k = search(ref[i]);</pre>	char c;	args);
printf("Total Page	if (k == -1)	fh = fopen(fn, "r");	if (strcmp(args[0],
Faults: %d\n", faults);	{	if (fh == NULL)	"count") == 0)
}	sp = get_Iru();	{	

count(args[2],	for (i = 0; i < m; i++)	k = search(ref[i]);	#include <stdlib.h></stdlib.h>
args[1][0]);	printf("%3d",	if (k == -1)	#include <unistd.h></unistd.h>
else	ref[i]);	{	#include <string.h></string.h>
{	printf("\n\n");	frames[sp] =	void
pid = fork();	for (i = 0; i < n; i++)	ref[i];	make_toks(char *s,
if (pid > 0)	{	count[sp]++;	char *tok[])
wait();	for $(j = 0; j < m;$	faults++;	{
else	j++)	sp++;	int i = 0;
{	{	for (j = 0; j < n;	char *p;
if	if (mem[i][j])	j++)	p = strtok(s, " ");
(execvp(args[0], args)	printf("%3d",		while (p != NULL)
== -1)	mem[i][j]);	frames[j];	{
, printf("Bad	else	}	tok[i++] = p;
command.\n");	printf(" ");	else	p = strtok(NULL, "
}	}	count[k]++;	");
}	printf("\n");	}	,,, }
}	}	sp = 0;	tok[i] = NULL;
return 0;	printf("Total Page	for (; i < m; i++)	}
}	Faults: %d\n", faults);	{	void search(char *fn,
J	}	k = search(ref[i]);	char op, char *pattern)
	int search(int pno)	if (k == -1)	{
	s	11 (K1) 5	ι FILE *fh;
slip 4	ા int i;	ι sp =	int count = 0;
1->	for (i = 0; i < n; i++)	get_mfu(sp);	char line[80];
#include <stdio.h></stdio.h>	for (1 = 0, 1 < 11, 1++)	frames[sp] =	fh = fopen(fn, "r");
#define MAX 20	if (frames[i] ==	ref[i];	if (fh == NULL)
int frames[MAX],	·	count[sp] = 1;	(III == NOLL)
ref[MAX],	pno)	faults++;	ι printf("File %s not
mem[MAX][MAX],	return i;	sp = (sp + 1) %	found.\n", fn);
faults,	roturn 1.	, , , ,	•
·	return -1;	n;	return;
sp, m, n,	}	for $(j = 0; j < n;$	}
count[MAX];	int get_mfu(int sp)	j++)	while (fgets(line, 80,
void accept()	inti may i may -	mem[j][i] =	fh) != NULL)
limb is	int i, max_i, max = -	frames[j];	{ :f /atuatu/line
int i;	9999;	}	if (strstr(line,
printf("Enter no.of	i = sp;	else	pattern) != NULL)
frames:");	do	count[k]++;	1
scanf("%d", &n);	{	}	count++;
printf("Enter no.of	if (count[i] > max)	}	if (op == 'a')
references:");	{		printf("%s",
scanf("%d", &m);	max = count[i];	int main()	line);
printf("Enter	max_i = i;	{	}
reference string:\n");	}	accept();	}
for (i = 0; i < m; i++)	i = (i + 1) % n;	mfu();	fclose(fh);
{	} while (i != sp);	disp();	if (op == 'c')
printf("[%d]=", i);	return max_i;	return 0;	printf("Number of
scanf("%d",	}	}	occurrences: %d\n",
&ref[i]);	void mfu()	_	count);
}	{	2->	}
}	int i, j, k;	#include <sys types.h=""></sys>	int main()
void disp()	for (i = 0; i < m &&	#include <sys stat.h=""></sys>	{
{	sp < n; i++)	#include <fcntl.h></fcntl.h>	char buff[80],
int i, j;	{	#include <stdio.h></stdio.h>	*args[10];

int pid;	scanf("%d",	if (flag2 == 0)	frames[pos] =
while (1)	<pre>&no_of_pages);</pre>	{	pages[i];
{		flag3 = 0;	faults++;
printf("myshell\$	printf("Enter page		}
");	reference string: ");	for (j = 0; j <	printf("\n");
fflush(stdin);		no_of_frames; ++j)	
fgets(buff, 80,	for (i = 0; i <	{	for (j = 0; j <
stdin);	no_of_pages; ++i)	temp[j] = -1;	no_of_frames; ++j)
buff[strlen(buff) -	{	,	_ {
1] = '\0';	scanf("%d",	for (k = i + 1; k	printf("%d\t",
make_toks(buff,	&pages[i]);	< no_of_pages; ++k)	frames[j]);
args);	}	{	}
if (strcmp(args[0],	,	if (frames[j]	}
"search") == 0)	for (i = 0; i <	== pages[k])	j
search(args[2],	no_of_frames; ++i)	pages[k])	printf("\n\nTotal
args[1][0], args[3]);	110_01_11 at 11e3, 111)	ι temp[j] =	Page Faults = %d",
else	frames[i] = -1;	k;	faults);
ſ	1 ames[i] = -1,	break;	rauits _j ,
\ nid = fork/\	ſ	n eak,	raturn O
pid = fork();	for /: - 0. : <	}	return 0;
if (pid > 0)	for (i = 0; i <	}	}
wait();	no_of_pages; ++i)	}	2->
else	{ fla=1, fla=2, 0;	ford: Octob	#include <sys types.h=""></sys>
{	flag1 = flag2 = 0;	for (j = 0; j <	#include <sys stat.h=""></sys>
if	5 (1 0 1	no_of_frames; ++j)	#include <fcntl.h></fcntl.h>
(execvp(args[0], args)	for (j = 0; j <	{	#include <stdio.h></stdio.h>
== -1)	no_of_frames; ++j)	if (temp[j] == -	#include <stdlib.h></stdlib.h>
printf("Bad	{	1)	#include <unistd.h></unistd.h>
command.\n");	if (frames[j] ==	{	#include <string.h></string.h>
}	pages[i])	pos = j;	void
}	{	flag3 = 1;	make_toks(char *s,
}	flag1 = flag2 =	break;	char *tok[])
return 0;	1;	}	{
}	break;	}	int i = 0;
	}		char *p;
slip 5	}	if (flag3 == 0)	p = strtok(s, " ");
1->		{	while (p != NULL)
#include <stdio.h></stdio.h>	if (flag1 == 0)	max =	{
int main()	{	temp[0];	tok[i++] = p;
{	for (j = 0; j <	pos = 0;	p = strtok(NULL, "
int no_of_frames,	no_of_frames; ++j)		");
no_of_pages,	{	for (j = 1; j <	}
frames[10], pages[30],	if (frames[j]	no_of_frames; ++j)	tok[i] = NULL;
temp[10], flag1, flag2,	== -1)	{	}
flag3, i, j,	{	if (temp[j] >	void search(char *fn,
k, pos, max, faults	faults++;	max)	char op, char *pattern)
= 0;	frames[j] =	. {	{
printf("Enter	pages[i];	max =	FILE *fh;
number of frames: ");	flag2 = 1;	temp[j];	int count = 0;
scanf("%d",	break;	pos = j;	char line[80];
&no_of_frames);	}	}	fh = fopen(fn, "r");
	}	}	if (fh == NULL)
printf("Enter	}	}	{
number of pages: ");	,	•	•

printf("File %s not	fgets(buff, 80,	scanf("%d",	int i, j, k;
found.\n", fn);	stdin);	&ref[i]);	for (i = 0; i < m &&
return;	buff[strlen(buff) -	}	sp < n; i++)
}	1] = '\0';	}	{
if (op == 'f')	make_toks(buff,	void disp()	
{	args);	{	k = search(ref[i]);
	if (strcmp(args[0],	int i, j;	if (k == -1)
while (fgets(line,	"search") == 0)	for (i = 0; i < m; i++)	{
80, fh) != NULL)	search(args[2],	printf("%3d",	frames[sp] =
{	args[1][0], args[3]);	ref[i]);	ref[i];
if (strstr(line,	else	printf("\n\n");	time[sp] = i;
pattern) != NULL)	{	for (i = 0; i < n; i++)	faults++;
{	pid = fork();	{	sp++;
	if (pid > 0)	for $(j = 0; j < m;$	for $(j = 0; j < n;$
printf("%s",	wait();	j++)	j++)
line);	else	{	mem[j][i] =
break;	{	if (mem[i][j])	frames[j];
}	if	printf("%3d",	}
}	(execvp(args[0], args)	mem[i][j]);	else
}	== -1)	else	time[k] = i;
else	printf("Bad	printf(" ");	}
{	command.\n");	}	for (i = 0; i < m; i++)
	}	printf("\n");	{
while (fgets(line,	}	}	k = search(ref[i]);
80, fh) != NULL)	}	printf("Total Page	if (k == -1)
{	return 0;	Faults: %d\n", faults);	{
if (strstr(line,	}	}	sp = get_mru();
pattern) != NULL)	•	int search(int pno)	frames[sp] =
{	slip 6	{	ref[i];
count++;	1->	int i;	time[sp] = i;
if (op == 'a')	#include <stdio.h></stdio.h>	for (i = 0; i < n; i++)	faults++;
printf("%s",	#define MAX 20	{	for (j = 0; j < n;
line);	int frames[MAX],	if (frames[i] ==	j++)
}	ref[MAX],	pno)	mem[j][i] =
}	mem[MAX][MAX],	return i;	frames[j];
if (op == 'c')	faults,	}	}
printf("Number	sp, m, n,	return -1;	else
of occurrences: %d\n",	count[MAX],	}	time[k] = i;
count);	time[MAX];	int get_mru()	}
}	void accept()	{	}
fclose(fh);	{	int i, max_i, max = 0;	int main()
}	int i;	for (i = 0; i < n; i++)	{
int main()	printf("Enter no.of	{	accept();
{	frames:");	if (time[i] > max)	mru();
char buff[80],	scanf("%d", &n);	{	disp();
*args[10];	printf("Enter no.of	max = time[i];	return 0;
int pid;	references:");	max_i = i;	}
while (1)	scanf("%d", &m);	}	•
{	printf("Enter	}	2->
printf("myshell\$	reference string:\n");	return max_i;	#include <sys types.h=""></sys>
");	for (i = 0; i < m; i++)	}	#include <sys stat.h=""></sys>
fflush(stdin);	{	void mru()	#include <fcntl.h></fcntl.h>
	printf("[%d]=", i);	{	#include <stdio.h></stdio.h>
	F([,00]),))	·	

#include <stdlib.h></stdlib.h>	while (fgets(line,	}	flag1 = flag2 =
#include <unistd.h></unistd.h>	80, fh) != NULL)	return 0;	1;
#include <string.h></string.h>	{	}	break;
void	if (strstr(line,		}
make_toks(char *s,	pattern) != NULL)		}
char *tok[])	{	slip 7	
{	count++;	1->	if (flag1 == 0)
int i = 0;	if (op == 'a')	#include <stdio.h></stdio.h>	{
char *p;	printf("%s",	int main()	for (j = 0; j <
p = strtok(s, " ");	line);	{	no_of_frames; ++j)
while (p != NULL)	}	int no_of_frames,	_ {
{	}	no_of_pages,	if (frames[j]
tok[i++] = p;	if (op == 'c')	frames[10], pages[30],	== -1)
p = strtok(NULL, "	printf("Number	temp[10], flag1, flag2,	{
");	of occurrences: %d\n",	flag3, i, j, k, pos, max,	faults++;
}	count);	faults = 0;	frames[j] =
tok[i] = NULL;	}	printf("Enter	pages[i];
}	fclose(fh);	number of frames: ");	flag2 = 1;
void search(char *fn,	}	scanf("%d",	break;
char op, char *pattern)	int main()	&no_of_frames);	}
{	{	"	}
FILE *fh;	char buff[80],	printf("Enter	}
int count = 0;	*args[10];	number of pages: ");	,
char line[80];	int pid;	scanf("%d",	if (flag2 == 0)
fh = fopen(fn, "r");	while (1)	&no_of_pages);	{
if (fh == NULL)	{		flag3 = 0;
{	printf("myshell\$	printf("Enter page	
printf("File %s not	");	reference string: ");	for (j = 0; j <
found.\n", fn);	fflush(stdin);	1 6 7 6 7 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7	no_of_frames; ++j)
return;	fgets(buff, 80,	for (i = 0; i <	{
}	stdin);	no_of_pages; ++i)	temp[j] = -1;
if (op == 'f')	buff[strlen(buff) -	{	
{	1] = '\0';	scanf("%d",	for (k = i + 1; k
while (fgets(line,	make_toks(buff,	&pages[i]);	< no_of_pages; ++k)
80, fh) != NULL)	args);	}	{
{	if (strcmp(args[0],	,	if (frames[j]
if (strstr(line,	"search") == 0)	for (i = 0; i <	== pages[k])
pattern) != NULL)	search(args[2],	no_of_frames; ++i)	{ pages[v])
{	args[1][0], args[3]);	{	temp[j] =
// Print the	else	frames[i] = -1;	k;
line and exit the loop	{	}	break;
printf("%s",	pid = fork();	,	}
line);	if (pid > 0)	for (i = 0; i <	,
break;	wait();	no_of_pages; ++i)	}
}	else	{	J
}	{	flag1 = flag2 = 0;	for (j = 0; j <
}	if	11481 11482 0,	no_of_frames; ++j)
else	(execvp(args[0], args)	for (j = 0; j <	{
{	== -1)	no_of_frames; ++j)	۱ if (temp[j] == -
۱ // Continue with	printf("Bad	{	1)
the existing search	command.\n");	۱ if (frames[j] ==	±,
implementation	}	pages[i])	ι pos = j;
implementation	}	{ hages[i])	flag3 = 1;
	J	ι	11455 - 1,

break;	make_toks(char *s,	printf("%s",	#define MAX 20
}	char *tok[])	line);	int frames[MAX],
}	{	}	ref[MAX],
	int i = 0;	}	mem[MAX][MAX],
if (flag3 == 0)	char *p;	if (op == 'c')	faults, sp, m, n,
{	p = strtok(s, " ");	printf("Number	time[MAX];
max =	while (p != NULL)	of occurrences: %d\n",	void accept()
temp[0];	{	count);	{
pos = 0;	tok[i++] = p;	}	int i;
•	p = strtok(NULL, "	fclose(fh);	printf("Enter the
for (j = 1; j <	");	}	number of frames: ");
no_of_frames; ++j)	}	int main()	scanf("%d", &n);
{	tok[i] = NULL;	{	printf("Enter the
if (temp[j] >	}	char buff[80],	number of references:
max)	void search(char *fn,	*args[10];	");
. {	char op, char *pattern)	int pid;	scanf("%d", &m);
max =	{	while (1)	printf("Enter the
temp[j];	FILE *fh;	{	reference string:\n");
pos = j;	int count = 0;	printf("myshell\$	for (i = 0; i < m; i++)
}	char line[80];	");	{
}	fh = fopen(fn, "r");	fflush(stdin);	printf("[%d] = ", i);
}	if (fh == NULL)	fgets(buff, 80,	scanf("%d",
frames[pos] =	{	stdin);	&ref[i]);
pages[i];	printf("File %s not	buff[strlen(buff) -	}
faults++;	found.\n", fn);	1] = '\0';	}
}	return;	make_toks(buff,	void disp()
j	}	args);	{
printf("\n");	if (op == 'f')	if (strcmp(args[0],	int i, j;
princi((ii),	((Sp .)	"search") == 0)	for (i = 0; i < m; i++)
for (j = 0; j <	while (fgets(line,	search(args[2],	printf("%3d",
no_of_frames; ++j)	80, fh) != NULL)	args[1][0], args[3]);	ref[i]);
{	{	else	printf("\n\n");
printf("%d\t",	if (strstr(line,	Į.	for (i = 0; i < n; i++)
frames[j]);	pattern) != NULL)	pid = fork();	{
}	{	if (pid > 0)	for (j = 0; j < m;
3	// Print the	wait();	j++)
,	line and exit the loop	else	1
printf("\n\nTotal	printf("%s",	{	ر if (mem[i][j])
Page Faults = %d",	line);	if	printf("%3d",
faults);	break;	(execvp(args[0], args)	mem[i][j]);
	}	== -1)	else
return 0;	}	printf("Bad	printf(" ");
1	}	command.\n");)
ı	else	}	printf("\n");
2->	{	1)
#include <sys types.h=""></sys>	while (fgets(line,	}	printf("Total Page
#include <sys stat.h=""></sys>	80, fh) != NULL)	return 0;	Faults: %d\n", faults);
#include <fcntl.h></fcntl.h>	{	}	}
#include <stdio.h></stdio.h>	ı if (strstr(line,	J	int search(int pno)
#include <stdlib.h></stdlib.h>	pattern) != NULL)		{
#include <unistd.h></unistd.h>	{	slip 8	ເ int i;
#include <string.h></string.h>	ι count++;	1->	for (i = 0; i < n; i++)
void	if (op == 'a')	#include <stdio.h></stdio.h>	{
VOIG	11 (OP u)	minerade State.	ι

if (frames[i] ==	time[sp] = i;	{	fgets(buff, 80,
pno)	count[sp] = 1;	printf("File %s not	stdin);
return i;	faults++;	found.\n", fn);	buff[strlen(buff) -
}	for $(j = 0; j < n;$	return;	1] = '\0';
return -1;	j++)	}	make_toks(buff,
}	mem[j][i] =	if (op == 'f')	args);
void Ifu()	frames[j];	{	if (strcmp(args[0],
{	}	while (fgets(line,	"search") == 0)
int i, j, k, count[20];	else	80, fh) != NULL)	search(args[2],
for (i = 0; i < m &&	{	{	args[1][0], args[3]);
sp < n; i++)	time[k] = i;	if (strstr(line,	else
{	count[k]++;	pattern) != NULL)	{
k = search(ref[i]);	}	{	pid = fork();
if (k == -1)	}	// Print the	if (pid > 0)
{	}	line and exit the loop	29345436
frames[sp] =	int main()	printf("%s",	wait();
ref[i];	{	line);	else
time[sp] = 1;	accept()	break;	{
count[sp] = 1;	lfu();	}	if
faults++;	disp();	}	(execvp(args[0], args)
sp++;	return 0;	}	== -1)
for (j = 0; j < n;	}	else	printf("Bad
j++)		{	command.\n");
mem[j][i] =	2->	while (fgets(line,	}
frames[j];	#include <sys types.h=""></sys>	80, fh) != NULL)	}
}	#include <sys stat.h=""></sys>	{	}
else	#include <fcntl.h></fcntl.h>	if (strstr(line,	return 0;
{	#include <stdio.h></stdio.h>	pattern) != NULL)	}
time[k] = i;	#include <stdlib.h></stdlib.h>	{	
count[k]++;	#include <unistd.h></unistd.h>	count++;	
}	#include <string.h></string.h>	if (op == 'a')	
}	void make_toks(char	printf("%s",	slip 9
for (; i < m; i++)	*s, char *tok[])	line);	1->
{	{	}	#include <stdio.h></stdio.h>
k = search(ref[i]);	int i = 0;	}	#define MAX 20
if (k == -1)	char *p;	if (op == 'c')	int frames[MAX],
{	p = strtok(s, " ");	printf("Number	ref[MAX],
int min_i = 0,	while (p != NULL)	of occurrences: %d\n",	mem[MAX][MAX],
min = 9999;	{	count);	faults, sp, m, n;
for $(j = 0; j < n;$	tok[i++] = p;	}	void accept()
j++)	p = strtok(NULL, "	fclose(fh);	{
{	");	}	int i;
if (count[j] <	}	int main()	printf("Enter no.of
min)	tok[i] = NULL;	{ 	frames:");
{	}	char buff[80],	scanf("%d", &n);
min =	void search(char *fn,	*args[10];	printf("Enter no.of
count[j];	char op, char *pattern)	int pid;	references:");
min_i = j; າ	([]] [*fh.	while (1)	scanf("%d", &m);
) l	FILE *fh;	nrin+f/"muchall¢	printf("Enter
sn - min i:	int count = 0; char line[80];	printf("myshell\$ ");	reference string:\n"); for (i = 0; i < m; i++)
sp = min_i; frames[sp] =	thar line[80]; fh = fopen(fn, "r");); fflush(stdin);	101 (1 - 0, 1 < 111; 1++) }
ref[i];	if (fh == NULL)	musii(stuiii),	ι printf("[%d]=", i);
יכינין,	11 (111 NOLL)		ριπτιί [/ou]- ,1),

scanf("%d",	mem[j][i] =	// Search for the	make_toks(buff,
&ref[i]);	frames[j];	first occurrence of the	args);
}	}	pattern	if (strcmp(args[0],
}	}	while (fgets(line,	"search") == 0)
void disp()	}	80, fh) != NULL)	search(args[2],
{	int main()	{	args[1][0], args[3]);
int i, j;	{	if (strstr(line,	else
for (i = 0; i < m; i++)	accept();	pattern) != NULL)	{
printf("%3d",	fifo();	{	pid = fork();
ref[i]);	disp();	// Print the	if (pid > 0)
printf("\n\n");	return 0;	line and exit the loop	wait();
for (i = 0; i < n; i++)	}	printf("%s",	else
{ • • • • • • • • • • • • • • • • • • •		line);	{
for $(j = 0; j < m;$	2->	break;	if
j++)	#include <sys types.h=""></sys>	}	(execvp(args[0], args)
{	#include <sys stat.h=""></sys>	, }	== -1)
if (mem[i][j])	#include <fcntl.h></fcntl.h>	}	printf("Bad
printf("%3d",	#include <stdio.h></stdio.h>	else	command.\n");
mem[i][j]);	#include <stdlib.h></stdlib.h>	{	}
else	#include <unistd.h></unistd.h>	while (fgets(line,	}
printf(" ");	#include <string.h></string.h>	80, fh) != NULL)	}
} nrin+f/"\ n"\.	void	if (stretr/line	return 0;
printf("\n");	make_toks(char *s, char *tok[])	if (strstr(line, pattern) != NULL)	}
f printf("Total Page	s tok[])	patterny := NOLL)	
Faults: %d\n", faults);	int i = 0;	ι count++;	slip 10
}	char *p;	if (op == 'a')	1->
int search(int pno)	p = strtok(s, " ");	printf("%s",	#include <stdio.h></stdio.h>
{	while (p != NULL)	line);	#define MAX 20
int i;	{	}	int frames[MAX],
for (i = 0; i < n; i++)	tok[i++] = p;	}	ref[MAX],
{	p = strtok(NULL, "	if (op == 'c')	mem[MAX][MAX],
if (frames[i] ==	");	printf("Number	faults, sp, m, n;
pno)	}	of occurrences: %d\n",	void accept()
return i;	tok[i] = NULL;	count);	{
}	}	}	int i;
return -1;	void search(char *fn,	fclose(fh);	printf("Enter no.of
}	char op, char *pattern)	}	frames:");
void fifo()	{	int main()	scanf("%d", &n);
{	FILE *fh;	{	printf("Enter no.of
int i, j;	int count = 0;	char buff[80],	references:");
for (i = 0; i < m; i++)	char line[80];	*args[10];	scanf("%d", &m);
{	fh = fopen(fn, "r");	int pid;	printf("Enter
if (search(ref[i]) ==	if (fh == NULL)	while (1)	reference string:\n");
-1)	{	{	for (i = 0; i < m; i++)
{	printf("File %s not	printf("myshell\$	{
frames[sp] =	found.\n", fn);	");	printf("[%d]=", i);
ref[i];	return;	fflush(stdin);	scanf("%d",
sp = (sp + 1) %	} :f /p	fgets(buff, 80,	&ref[i]);
n;	if (op == 'f')	stdin);	}
faults++;	{	buff[strlen(buff) -) void disp()
for (j = 0; j < n;		1] = '\0';	void disp()
j++)			ι

int i, j;	{	{	else
for (i = 0; i < m; i++)	accept();	case 'f':	{
printf("%3d",	fifo();	// Display the	if
ref[i]);	disp();	filename	(execvp(args[0], args)
printf("\n\n");	return 0;	printf("%s\n",	== -1)
for (i = 0; i < n; i++)	}	entry->d_name);	printf("Bad
{	J	break;	command.\n");
for (j = 0; j < m;	2->	case 'n':	Command. (ii),
j++)	#include <unistd.h></unistd.h>	count++;) l
)++)	#include <sys types.h=""></sys>	break;	}
if (mom[i][i])	#include <sys types.ii=""> #include <sys stat.h=""></sys></sys>	case 'i':	roturn O
if (mem[i][j])	#include <sys stat.n=""> #include <fcntl.h></fcntl.h></sys>		return 0;
printf("%3d",		printf("%s:	}
mem[i][j]);	#include <stdio.h></stdio.h>	%lu\n", entry-	
else	#include <stdlib.h></stdlib.h>	>d_name, entry-	1: 44
printf(" ");	#include <string.h></string.h>	>d_ino);	slip 11
}	#include <dirent.h></dirent.h>	break;	1->
printf("\n");	void	}	#include <stdio.h></stdio.h>
}	make_toks(char *s,	}	#define MAX 20
printf("Total Page	char *tok[])		int frames[MAX],
Faults: %d\n", faults);	{	closedir(dir);	ref[MAX],
}	int i = 0;	if (op == 'n')	mem[MAX][MAX],
int search(int pno)	char *p;	{	faults, sp, m, n,
{	p = strtok(s, " ");	printf("Number of	time[MAX];
int i;	while (p != NULL)	entries: %d\n", count);	void accept()
for (i = 0; i < n; i++)	{	}	{
{	tok[i++] = p;	}	int i;
if (frames[i] ==	p = strtok(NULL, "	int main()	printf("Enter the
pno)	");	{	number of frames: ");
return i;	}	char buff[80],	scanf("%d", &n);
}	tok[i] = NULL;	*args[10];	printf("Enter the
return -1;	}	int pid;	number of references:
}	void list(char	while (1)	");
void fifo()	*dirname, char op)	{	scanf("%d", &m);
{	{	printf("myshell\$	printf("Enter the
int i, j;	DIR *dir;	");	reference string:\n");
for (i = 0; i < m; i++)	struct dirent *entry;	fflush(stdin);	for (i = 0; i < m; i++)
f	int count = 0;	fgets(buff, 80,	{
if (search(ref[i]) ==	dir =	stdin);	printf("[%d] = ", i);
		• •	
-1)	opendir(dirname);	buff[strlen(buff) -	scanf("%d",
framestant -	if (dir == NULL)	1] = '\0';	&ref[i]);
frames[sp] =	{	make_toks(buff,	,
ref[i];	printf("Directory	args);	}
sp = (sp + 1) %	%s not found.\n",	if (strcmp(args[0],	void disp()
n;	dirname);	"list") == 0)	{
faults++;	return;	{	int i, j;
for (j = 0; j < n;	}	list(args[2],	for (i = 0; i < m; i++)
j++)	// Iterate through	args[1][0]);	printf("%3d",
mem[j][i] =	the entries in the	}	ref[i]);
frames[j];	directory	else	printf("\n\n");
}	while ((entry =	{	for (i = 0; i < n; i++)
}	readdir(dir)) != NULL)	pid = fork();	{
}	{	if (pid > 0)	for $(j = 0; j < m;$
int main()	switch (op)	wait();	j++)

{	{	int i = 0;	}
if (mem[i][j])	int min_i = 0,	char *p;	}
printf("%3d",	min = 9999;	p = strtok(s, " ");	int main()
mem[i][j]);	for $(j = 0; j < n;$	while (p != NULL)	{
else	j++)	{	char buff[80],
printf(" ");	{	tok[i++] = p;	*args[10];
}	if (count[j] <	p = strtok(NULL, "	int pid;
printf("\n");	min)	");	while (1)
}	, {	}	{
printf("Total Page	min =	tok[i] = NULL;	printf("myshell\$
Faults: %d\n", faults);	count[j];	}	");
}	min_i = j;	void list(char	fflush(stdin);
int search(int pno)	}	*dirname, char op)	fgets(buff, 80,
{	}	{	stdin);
int i;	sp = min_i;	DIR *dir;	buff[strlen(buff) -
for (i = 0; i < n; i++)	frames[sp] =	struct dirent *entry;	1] = '\0';
{	ref[i];	int count = 0;	make_toks(buff,
if (frames[i] ==	time[sp] = i;	// Open the	args);
pno)	count[sp] = 1;	directory	if (strcmp(args[0],
return i;	faults++;	dir =	"list") == 0)
}	for $(j = 0; j < n;$	opendir(dirname);	{
return -1;	j++)	if (dir == NULL)	list(args[2],
}	mem[j][i] =	{	args[1][0]);
void Ifu()	frames[j];	printf("Directory	}
{	}	%s not found.\n",	else
int i, j, k, count[20];	else	dirname);	{
for (i = 0; i < m &&	{	return;	pid = fork();
sp < n; i++)	time[k] = i;	}	if (pid > 0)
{	count[k]++;	while ((entry =	wait();
k = search(ref[i]);	}	readdir(dir)) != NULL)	else
if (k == -1)	}	{	{
{	}	switch (op)	if
frames[sp] =	int main()	{	(execvp(args[0], args)
ref[i];	{	case 'f':	== -1)
time[sp] = 1;	accept();	printf("%s\n",	printf("Bad
count[sp] = 1;	lfu();	entry->d_name);	command.\n");
faults++;	disp();	break;	}
sp++;	return 0;	case 'n':	}
for $(j = 0; j < n;$	}	count++;	}
j++)		break;	return 0;
mem[j][i] =	2->	case 'i':	}
frames[j];	#include <unistd.h></unistd.h>	printf("%s:	
}	#include <sys types.h=""></sys>	%lu\n", entry-	
else	#include <sys stat.h=""></sys>	>d_name, entry-	slip 12
{	#include <fcntl.h></fcntl.h>	>d_ino);	1->
time[k] = i;	#include <stdio.h></stdio.h>	break;	
count[k]++;	#include <stdlib.h></stdlib.h>	}	#include <stdio.h></stdio.h>
}	#include <string.h></string.h>	}	#define MAX 20
}	#include <dirent.h></dirent.h>	closedir(dir);	int
for (; i < m; i++)	void	if (op == 'n')	frames[MAX],ref[MAX]
{	make_toks(char *s,	{	,mem[MAX][MAX],faul
k = search(ref[i]);	char *tok[])	printf("Number of	ts,
if (k == -1)	{	entries: %d\n", count);	sp,m,n,time[MAX];

void accept()	ſ	ì	ı
void accept()	ι int i, min_i, min =	int main()	ι case 'f':
ι int i;	9999;	s	// Display the
printf("Enter no.of	for (i = 0; i < n; i++)	l accont():	filename
•	101 (1 = 0, 1 < 11, 1++)	accept();	
frames:");	if (+im of il < min)	lru();	printf("%s\n",
scanf("%d", &n);	if (time[i] < min)	disp();	entry->d_name);
printf("Enter no.of	{ 	return 0;	break;
references:");	min = time[i];	}	case 'n':
scanf("%d", &m);	min_i = i;		// Increment
printf("Enter	}	2->	the count
reference string:\n");	}		count++;
for (i = 0; i < m; i++)	return min_i;	#include <unistd.h></unistd.h>	break;
{	}	#include <sys types.h=""></sys>	case 'i':
printf("[%d]=", i);	void lru()	#include <sys stat.h=""></sys>	// Display the
scanf("%d",	{	#include <fcntl.h></fcntl.h>	filename and inode
&ref[i]);	int i, j, k;	#include <stdio.h></stdio.h>	number
}	for (i = 0; i < m &&	#include <stdlib.h></stdlib.h>	printf("%s:
}	sp < n; i++)	#include <string.h></string.h>	%lu\n", entry-
void disp()	{	#include <dirent.h></dirent.h>	>d_name, entry-
{	k = search(ref[i]);	void	>d_ino);
int i, j;	if (k == -1)	make_toks(char *s,	break;
for $(i = 0; i < m; i++)$	{	char *tok[])	}
printf("%3d",	frames[sp] =	{	}
ref[i]);	ref[i];	int i = 0;	
printf("\n\n");	time[sp] = i;	char *p;	closedir(dir);
for (i = 0; i < n; i++)	faults++;	p = strtok(s, " ");	
{	sp++;	while (p != NULL)	if (op == 'n')
for $(j = 0; j < m;$	for $(j = 0; j < n;$	{	{
j++)	j++)	tok[i++] = p;	printf("Number of
{	mem[j][i] =	p = strtok(NULL, "	entries: %d\n", count);
if (mem[i][j])	frames[j];	");	}
printf("%3d",	}	}	}
mem[i][j]);	else	tok[i] = NULL;	int main()
else	time[k] = i;	}	{
printf(" ");	}	void list(char	char buff[80],
}	for $(i = 0; i < m; i++)$	*dirname, char op)	*args[10];
printf("\n");	{	{	int pid;
}	k = search(ref[i]);	DIR *dir;	while (1)
printf("Total Page	if (k == -1)	struct dirent *entry;	{
Faults: %d\n", faults);	{	int count = 0;	printf("myshell\$
}	sp = get_Iru();	dir =	");
int search(int pno)	frames[sp] =	opendir(dirname);	fflush(stdin);
{	ref[i];	if (dir == NULL)	fgets(buff, 80,
int i;	time[sp] = i;	{	stdin);
for (i = 0; i < n; i++)	faults++;	printf("Directory	buff[strlen(buff) -
{	for (j = 0; j < n;	%s not found.\n",	1] = '\0';
if (frames[i] ==	j++)	dirname);	make_toks(buff,
pno)	, , mem[j][i] =	return;	args);
return i;	frames[j];	}	if (strcmp(args[0],
}	}	while ((entry =	"list") == 0)
return -1;	else	readdir(dir)) != NULL)	{
}	time[k] = i;	{	list(args[2],
int get_lru()	}	switch (op)	args[1][0]);
6	,	(//////////////	O-1-11-1//

}	fgets(buff, 80,		scanf("%d",
else	stdin);	printf("Error opening	&job[i].at);
{	buff[strlen(buff) -	file %s.\n", args[2]);	printf("Enter the
pid = fork();	1] = '\0';	return 1;	burst time of the job:
if (pid > 0)	make_toks(buff,	}	");
wait();	args);	char line[80];	scanf("%d",
else	if (strcmp(args[0],	while	&job[i].bt);
{	"typeline") == 0)	(fgets(line, 80, fp) !=	job[i].tbt =
if	{	NULL)	job[i].bt;
(execvp(args[0], args)	if (args[1][0] ==	printf("%s",	printf("\n\n");
== -1)	'+')	line);	}
printf("Bad	{	fclose(fp);	}
command.\n");	// Read the	}	
}	line count	else	// to sort the
}	int n =	{	processes by arriaval
}	atoi(args[1] + 1);	printf("Invalid	time
return 0;	// Open the	option.\n");	sort()
}	file	}	{
	FILE *fp =	}	struct job temp;
	fopen(args[2], "r");	}	for (i = 0; i < n; i++)
slip 13	if $(fp == NULL)$	}	{
1->	{		for $(j = i + 1; j < n;$
#include <sys types.h=""></sys>		2->	j++)
#include <sys stat.h=""></sys>	printf("Error opening	#include <stdio.h></stdio.h>	{
#include <fcntl.h></fcntl.h>	file %s.\n", args[2]);	#include <string.h></string.h>	if (job[i].at >
#include <stdio.h></stdio.h>	return 1;	struct job	job[j].at)
#include <stdlib.h></stdlib.h>	}	{	{
#include <unistd.h></unistd.h>	// Read and	char name[20];	temp = job[i];
#include <string.h></string.h>	print the first n lines	int at, bt, ct, tat, wt,	job[i] = job[j];
void make_toks(char	char line[80];	st, tbt;	job[j] = temp;
*s, char *tok[])	int count = 0;	} job[10];	}
{	while (count <	int n, i, j, tq;	}
int i = 0;	n && fgets(line, 80, fp)	float avg_tat = 0;	}
char *p;	!= NULL)	float avg_wt = 0;	}
p = strtok(s, " ");	{	// to accept the info	// to calculate the tat n
while (p != NULL)	printf("%s",	about processes	wt
{	line);	take_input()	void process()
tok[i++] = p;	count++;	{	{
p = strtok(NULL, "	}	printf("Enter the no	int jno = 0, cnt = 0,
");	// Close the	of jobs : ");	time = job[0].at;
}	file	scanf("%d", &n);	
tok[i] = NULL;	fclose(fp);	printf("Enter the	printf("\n********
}	}	time Quantum: ");	Gantt
int main()	else if	scanf("%d", &tq);	chart*******\n");
{	(strcmp(args[2], "-a")	for (i = 0; i < n; i++)	while (1)
char buff[80],	== 0)	{	{
*args[10];	{	printf("Enter the	if (job[jno].tbt !=
int pid;	// Open the	name of the job: ");	0)
while (1)	file	scanf("%s",	{
{	FILE *fp =	&job[i].name);	printf(" %d %s
printf("myshell\$	fopen(args[2], "r");	printf("Enter the	", time,
");	if (fp == NULL)	arrival time of the job :	job[jno].name);
fflush(stdin);	{	");	

if (job[jno].tbt	printf("\n%s %d	{	{
>= tq)	%d %d %d",	tok[i++] = p;	printf("%s",
{	job[i].name, job[i].at,	p = strtok(NULL, "	line);
job[jno].tbt =	job[i].bt, job[i].tat,	");	count++;
job[jno].tbt - tq;	job[i].wt);	}	}
time = time +	avg_tat = avg_tat	tok[i] = NULL;	// Close the
tq;	+ (float)(job[i].tat);	}	file
}	avg_wt =	int main()	fclose(fp);
else	(float)avg_wt +	{	}
{	(float)(job[i].wt);	char buff[80],	// If the '-a'
time = time +	}	*args[10];	option is specified
job[jno].tbt;	printf("\n	int pid;	else if
job[jno].tbt =	");	while (1)	(strcmp(args[2], "-a")
0;	printf("\nThe avg of	{	== 0)
}	the turn around time is	printf("myshell\$	{
printf("%d ",	%f", avg_tat / n);	");	// Open the
time);	printf("\nThe avg of	fflush(stdin);	file
if (job[jno].tbt	the waiting time is %f",	fgets(buff, 80,	FILE *fp =
== 0)	avg_wt / n);	stdin);	fopen(args[2], "r");
{	}	buff[strlen(buff) -	if (fp == NULL)
job[jno].ct =	main()	1] = '\0';	{
time;	{	make_toks(buff,	·
job[jno].tat =	take_input();	args);	printf("Error opening
time - job[jno].at;	process();	if (strcmp(args[0],	file %s.\n", args[2]);
job[jno].wt =	print_output();	"typeline") == 0)	return 1;
job[jno].tat -	for (i = 0; i < n; i++)	, {	}
job[jno].bt;	{	// Check if the	// Read and
cnt++;	job[i].tbt =	'+n' option is specified	print the entire file
}	job[i].bt = rand() % 10	if (args[1][0] ==	char line[80];
}	+ 1;	'+')	while
jno++;	_, job[i].at = job[i].ct	,	(fgets(line, 80, fp) !=
if (jno == n)	+ 2;	// Read the	NULL)
{	}	line count	printf("%s",
jno = 0;	process();	int n =	line);
}	print_output();	atoi(args[1] + 1);	// Close the
if (cnt == n)	}	// Open the	file
break;	j	file	fclose(fp);
}	slip 14	FILE *fp =	}
}	1->	fopen(args[2], "r");	else
// to print the output	#include <sys types.h=""></sys>	if (fp == NULL)	{
table	#include <sys stat.h=""></sys>	{	printf("Invalid
void print_output()	#include <fcntl.h></fcntl.h>	· ·	option.\n");
{	#include <stdio.h></stdio.h>	printf("Error opening	}
printf("\n\n");	#include <stdlib.h></stdlib.h>	file %s.\n", args[2]);	}
printf("\n	#include <unistd.h></unistd.h>	return 1;	}
");	#include <string.h></string.h>	\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	}
printf("\n pname	void make_toks(char	// Read and	J
at bt tat wt");	*s, char *tok[])	print the first n lines	2->
printf("\n	3, Chai (UK[]) {	char line[80];	#include <stdio.h></stdio.h>
");	ι int i = 0;	int count = 0;	#include <stdlib.h></stdlib.h>
for (i = 0; i < n; i++)	char *p;	while (count <	#include <string.h></string.h>
101 (1 - 0, 1 \ 11, 1++) {	• •	·	•
1	p = strtok(s, " ");	n && fgets(line, 80, fp)	typedef struct
	while (p != NULL)	!= NULL)	process_info

```
{
                                                                                                  } s[100], s1[100];
                                     int tat = p->ct - p-
                                >at;
  char pname[20];
                                                                          strcpy(name,
                                                                                                  int k;
                                                                                                  void sjfnp()
  int at, bt, ct, bt1;
                                     int wt = tat - p-
                                                                 p->pname);
  struct process_info
                                >bt;
                                                                          strcpy(p-
*next;
                                                                 >pname, q->pname);
                                                                                                    int prev = 0, n1 = 0;
} NODE;
                                                                                                    NODE *p;
                                     avg tat += tat;
                                                                          strcpy(q-
                                                                 >pname, name);
                                                                                                    while (n1 != n)
int n;
                                     avg_wt += wt;
NODE *first, *last;
                                                                                                    {
                                                                          t = p->at;
void accept info()
                                printf("%s\t%d\t%d\t%
                                                                           p->at = q->at;
                                                                                                      p = get sif();
                                d\t%d\t%d\n",
                                                                           q->at=t;
                                                                                                      if (p == NULL)
  NODE *p;
                                         p->pname, p-
  int i;
                                >at, p->bt, p->ct, tat,
                                                                           t = p -> bt;
                                                                                                         time++;
  printf("Enter no.of
                                wt);
                                                                           p->bt = q->bt;
                                                                                                         s[k].start = prev;
process:");
                                                                           q - bt = t;
  scanf("%d", &n);
                                                                           t = p->ct;
                                                                                                  strcpy(s[k].pname,
                                     p = p->next;
                                                                                                  "*");
  for (i = 0; i < n; i++)
                                   }
                                                                           p->ct = q->ct;
  {
                                   printf("Avg
                                                                           q->ct=t;
                                                                                                         s[k].end = time;
    p = (NODE)
                                TAT=%f\tAvg
                                                                                                         prev = time;
                                                                           t = p->bt1;
*)malloc(sizeof(NODE))
                                WT=%f\n'',
                                                                           p->bt1 = q-
                                                                                                         k++;
                                       avg_tat / n,
                                                                 >bt1;
                                                                                                      }
    printf("Enter
                                avg_wt / n);
                                                                           q->bt1=t;
                                                                                                      else
                                                                        }
process name:");
    scanf("%s", p-
                                void print_input()
                                                                        q = q->next;
                                                                                                         time += p->bt1;
>pname);
                                                                      }
                                                                                                         s[k].start = prev;
    printf("Enter
                                   NODE *p;
arrival time:");
                                   p = first;
                                                                      p = p->next;
                                                                                                  strcpy(s[k].pname, p-
    scanf("%d", &p-
                                                                   }
                                                                                                  >pname);
>at);
                                                                                                         s[k].end = time;
    printf("Enter first
                                printf("pname\tat\tbt\
                                                                                                         prev = time;
                                                                 int time;
CPU burst time:");
                                n");
                                                                 NODE *get_sjf()
                                                                                                         k++;
    scanf("%d", &p-
                                                                                                         p->ct = time;
                                   while (p != NULL)
                                                                 {
>bt);
                                   {
                                                                    NODE *p, *min_p =
                                                                                                         p->bt1 = 0;
                                                                 NULL;
                                                                                                         n1++;
                                                                    int min = 9999;
    p->bt1 = p->bt;
                                printf("%s\t%d\t%d\n"
    p->next = NULL;
                                                                    p = first;
                                                                                                       print_input();
    if (first == NULL)
                                         p->pname, p-
                                                                   while (p != NULL)
                                                                                                      sort();
                                                                                                    }
       first = p;
                                >at, p->bt1);
    else
                                                                      if (p->at <= time
                                                                                                  }
                                     p = p->next;
                                   }
                                                                 && p->bt1 != 0 &&
       last->next = p;
                                                                                                  void
                                                                        p->bt1 < min)
    last = p;
                                }
                                                                                                  print gantt chart()
  }
                                void sort()
}
                                {
                                                                        min = p->bt1;
                                                                                                    int i, j, m;
void print_output()
                                   NODE *p, *q;
                                                                        min_p = p;
                                                                                                    s1[0] = s[0];
                                   int t;
  NODE *p;
                                   char name[20];
                                                                                                    for (i = 1, j = 0; i < k;
                                                                      p = p->next;
  float avg tat = 0,
                                   p = first;
                                                                                                  i++)
avg_wt = 0;
                                   while (p->next !=
                                                                    return min_p;
                                                                                                    {
                                NULL)
printf("pname\tat\tbt\
                                   {
                                                                 struct gantt chart
                                                                                                  (strcmp(s[i].pname,
tct\ttat\twt\n");
                                     q = p->next;
                                                                                                  s1[j].pname) == 0
                                     while (q != NULL)
  p = first;
                                                                    int start;
                                                                                                         s1[j].end =
  while (p != NULL)
                                     {
                                                                    char pname[30];
                                                                                                  s[i].end;
  {
                                       if (p->at > q->at)
                                                                    int end;
                                                                                                      else
```

s1[++j] = s[i];	tok[i] = NULL;	{	int at, bt, ct, bt1;
}	}	printf("Number of	struct process_info
printf("%d",	void list(char	entries: %d\n", count);	*next;
s1[0].start);	*dirname, char op)	}	} NODE;
for (i = 0; i <= j; i++)	{	}	int n;
{	DIR *dir;	int main()	NODE *first, *last;
m = (s1[i].end -	struct dirent *entry;	{	void accept_info()
s1[i].start);	int count = 0;	char buff[80],	{
for (k = 0; k < m /	// Open the	*args[10];	NODE *p;
2; k++)	directory	int pid;	int i;
printf("-");	dir =	while (1)	•
		wille (1)	printf("Enter no.of
printf("%s",	opendir(dirname);	\ 	process:");
s1[i].pname);	if (dir == NULL)	printf("myshell\$	scanf("%d", &n);
for (k = 0; k < (m +	{	");	for (i = 0; i < n; i++)
1) / 2; k++)	printf("Directory	fflush(stdin);	{
printf("-");	%s not found.\n",	fgets(buff, 80,	p = (NODE
printf("%d",	dirname);	stdin);	*)malloc(sizeof(NODE))
s1[i].end);	return;	buff[strlen(buff) -	;
}	}	1] = '\0';	printf("Enter
}	// Iterate through	make_toks(buff,	process name:");
int main()	the entries in the	args);	scanf("%s", p-
{	directory	if (strcmp(args[0],	>pname);
accept_info();	while ((entry =	"list") == 0)	printf("Enter
sort();	readdir(dir)) != NULL)	{	arrival time:");
sjfnp();	{	list(args[2],	scanf("%d", &p-
print_output();	switch (op)	args[1][0]);	>at);
print_gantt_chart();	{	}	printf("Enter first
return 0;	case 'f':	else	CPU burst time:");
}	// Display the	{	scanf("%d", &p-
	filename	pid = fork();	>bt);
	printf("%s\n",	if (pid > 0)	p->bt1 = p->bt;
slip 15	entry->d_name);	wait();	
1->	break;	else	p->next = NULL;
#include <unistd.h></unistd.h>	case 'n':	{	if (first == NULL)
#include <sys types.h=""></sys>	// Increment	if	first = p;
#include <sys stat.h=""></sys>	the count	(execvp(args[0], args)	else
#include <fcntl.h></fcntl.h>	count++;	== -1)	last->next = p;
#include <stdio.h></stdio.h>	break;	printf("Bad	last = p;
#include <stdlib.h></stdlib.h>	case 'i':	command.\n");	}
#include <string.h></string.h>	// Display the	}	}
#include <dirent.h></dirent.h>	filename and inode	}	void print_output()
void make_toks(char	number	}	{
*s, char *tok[])	printf("%s:	return 0;	NODE *p;
{	%lu\n", entry-	}	float avg_tat = 0,
int i = 0;	>d_name, entry-	J	avg_wt = 0;
char *p;	>d_ino);	2->	uvg_wc = 0,
p = strtok(s, " ");	break;	#include <stdio.h></stdio.h>	printf("pname\tat\tbt\
while (p != NULL))	#include <stdlib.h></stdlib.h>	tct\ttat\twt\n");
νντιτία (ρ := INOLL)	,	#include <string.h></string.h>	p = first;
tok[i++] = p;	// Close the	typedef struct	while (p != NULL)
p = strtok(NULL, "		process_info	vviiiie (p :- NOLL)
· · · · · · · · · · · · · · · · · · ·	directory closedir(dir);	process_iiiio	int tot = n > ct n
"); \	if (op == 'n')	t char pname[20];	int tat = p->ct - p-
}	π (ορ π)	chai phame[20],	>at;

```
int k;
                                                                                                          s1[++j] = s[i];
    int wt = tat - p-
                                          strcpy(name,
                                                                  void sjfp()
>bt;
                                                                                                     }
                                 p->pname);
                                                                                                     printf("%d",
                                          strcpy(p-
                                                                    int prev = 0, n1 = 0;
                                 >pname, q->pname);
                                                                                                   s1[0].start);
    avg_tat += tat;
                                                                    NODE *p;
                                                                                                     for (i = 0; i \le j; i++)
    avg_wt += wt;
                                          strcpy(q-
                                                                    while (n1 != n)
                                 >pname, name);
printf("%s\t%d\t%d\t%
                                                                                                       m = (s1[i].end -
                                          t = p->at;
d\t%d\t%d\n'',
                                                                                                   s1[i].start);
                                          p->at = q->at;
                                                                      p = get_sjf();
                                                                                                       for (k = 0; k < m /
                                                                      if (p == NULL)
         p->pname, p-
                                          q->at=t;
>at, p->bt, p->ct, tat,
                                                                                                   2; k++)
                                                                                                          printf("-");
wt);
                                          t = p->bt;
                                                                         time++;
                                          p->bt = q->bt;
                                                                         s[k].start = prev;
                                                                                                       printf("%s",
    p = p->next;
                                          q->bt=t;
                                                                                                   s1[i].pname);
                                                                                                       for (k = 0; k < (m +
                                          t = p->ct;
                                                                  strcpy(s[k].pname,
                                                                  "*");
  printf("Avg
                                                                                                   1) / 2; k++)
                                          p->ct = q->ct;
TAT=%f\tAvg
                                                                                                          printf("-");
                                          q->ct=t;
                                                                         s[k].end = time;
                                                                                                       printf("%d",
WT=%f\n'',
                                          t = p -> bt1;
                                                                         prev = time;
                                                                                                   s1[i].end);
      avg_tat / n,
                                          p->bt1=q-
                                                                         k++;
avg_wt / n);
                                 >bt1;
                                                                      }
                                                                                                     }
                                                                      else
                                                                                                   }
                                          q->bt1=t;
void print_input()
                                                                                                   int main()
                                        q = q->next;
                                                                         time++;
  NODE *p;
                                     }
                                                                         s[k].start = prev;
                                                                                                     accept_info();
  p = first;
                                                                                                     sort();
                                                                  strcpy(s[k].pname, p-
                                                                                                     sjfp();
                                     p = p->next;
                                   }
                                                                  >pname);
                                                                                                     print_output();
printf("pname\tat\tbt\
                                 }
                                                                         s[k].end = time;
                                                                                                     print_gantt_chart();
n");
                                 int time;
                                                                         prev = time;
                                                                                                     return 0;
                                 NODE *get_sjf()
                                                                                                   }
  while (p != NULL)
                                                                         k++;
                                                                         p->ct = time;
                                   NODE *p, *min_p =
                                                                         p->bt1--;
printf("%s\t%d\t%d\n"
                                 NULL;
                                                                         if (p->bt1 == 0)
                                                                                                   slip 16
                                   int min = 9999;
                                                                           n1++;
                                                                                                   1->
                                   p = first;
                                                                      }
                                                                                                   #include <sys/types.h>
         p->pname, p-
>at, p->bt1);
                                   while (p != NULL)
                                                                      print_input();
                                                                                                   #include <sys/stat.h>
    p = p->next;
                                                                      sort();
                                                                                                   #include <fcntl.h>
  }
                                     if (p->at <= time
                                                                    }
                                                                                                   #include <stdio.h>
                                 && p->bt1 != 0 &&
                                                                                                   #include <stdlib.h>
                                                                  }
void sort()
                                                                                                   #include <unistd.h>
                                        p->bt1 < min)
                                                                  void
                                     {
                                                                  print_gantt_chart()
                                                                                                   #include <string.h>
  NODE *p, *q;
                                                                                                   void make_toks(char
                                        min = p->bt1;
  int t;
                                                                                                   *s, char *tok[])
                                        min_p = p;
                                                                    int i, j, m;
  char name[20];
                                     }
                                                                    s1[0] = s[0];
                                                                                                   {
                                     p = p->next;
  p = first;
                                                                                                     int i = 0;
                                                                    for (i = 1, j = 0; i < k;
  while (p->next !=
                                   }
                                                                                                     char *p;
NULL)
                                                                                                     p = strtok(s, " ");
                                                                  i++)
                                   return min_p;
                                                                                                     while (p != NULL)
  {
                                                                      if
    q = p->next;
                                 struct gantt_chart
                                                                                                     {
    while (q != NULL)
                                 {
                                                                  (strcmp(s[i].pname,
                                                                                                       tok[i++] = p;
                                   int start;
                                                                  s1[j].pname) == 0)
                                                                                                       p = strtok(NULL, "
                                                                                                   ");
       if (p->at > q->at)
                                   char pname[30];
                                                                         s1[j].end =
       {
                                   int end;
                                                                  s[i].end;
                                 } s[100], s1[100];
                                                                                                     tok[i] = NULL;
                                                                      else
```

}	printf("myshell\$	{	p->pname, p-
void count(char *fn,	");	p = (NODE	>at, p->bt, p->p, p->ct,
char op)	fflush(stdin);	*)malloc(sizeof(NODE))	tat, wt);
{	fgets(buff, 80,	;	
· FILE *fh;	stdin);	printf("Enter	p = p->next;
int cc = 0, wc = 0, lc =	buff[strlen(buff) -	process name:");	}
0;	1] = '\0';	scanf("%s", p-	printf("Avg
char c;	make_toks(buff,	>pname);	TAT=%f\tAvg
fh = fopen(fn, "r");	args);	printf("Enter	WT=%f\n",
if (fh == NULL)	if (strcmp(args[0],	arrival time:");	avg_tat / n,
{ ` ` ` ` ` ` ` ` `	"count") == 0)	scanf("%d", &p-	avg_wt / n);
printf("File %s not	count(args[2],	>at);	}
found.\n", fn);	args[1][0]);	printf("Enter first	void print_input()
return;	else	CPU burst time:");	{
}	{	scanf("%d", &p-	NODE *p;
while ((c = fgetc(fh))	pid = fork();	>bt);	p = first;
!= EOF)	if (pid > 0)	printf("Enter	F 7
{	wait();	priority:");	
if (c == ' ')	else	scanf("%d", &p-	printf("pname\tat\tbt\
wc++;	{	>p);	tp\n");
else if (c == '\n')	if	p->bt1 = p->bt;	while (p != NULL)
{	(execvp(args[0], args)	h h	{
wc++;	== -1)	p->next = NULL;	•
lc++;	printf("Bad	if (first == NULL)	printf("%s\t%d\t%d\t%
}	command.\n");	first = p;	d\n",
CC++;	}	else	p->pname, p-
}	}	last->next = p;	>at, p->bt1, p->p);
fclose(fh);	}	last = p;	p = p->next;
switch (op)	return 0;	}	}
{	}	}	}
case 'c':		<pre>void print_output()</pre>	void sort()
printf("No.of	Q .2	{	{
characters:%d\n", cc -	#include <stdio.h></stdio.h>	NODE *p;	NODE *p, *q;
1);	#include <stdlib.h></stdlib.h>	float avg_tat = 0,	int t;
break;	#include <string.h></string.h>	avg_wt = 0;	char name[20];
case 'w':	typedef struct		p = first;
printf("No.of	process_info	printf("pname\tat\tbt\	while (p->next !=
words:%d\n", wc);	{	tp\ttct\ttat\twt\n");	NULL)
break;	char pname[20];	p = first;	{
case 'I':	int at, bt, ct, bt1, p;	while (p != NULL)	q = p->next;
printf("No.of	struct process_info	{	while (q != NULL)
lines:%d\n", lc + 1);	*next;	int tat = p->ct - p-	{
break;	} NODE;	>at;	if (p->at > q->at)
}	int n;	int wt = tat - p-	{
}	NODE *first, *last;	>bt;	strcpy(name,
int main()	<pre>void accept_info()</pre>		p->pname);
{	{	avg_tat += tat;	strcpy(p-
char buff[80],	NODE *p;	avg_wt += wt;	>pname, q->pname);
*args[10];	int i;		strcpy(q-
int pid;	printf("Enter no.of	printf("%s\t%d\t%d\t%	>pname, name);
while (1)	process:");	$d\t%d\t%d\t$ ",	t = p->at;
{	scanf("%d", &n);		p->at = q->at;
	for (i = 0; i < n; i++)		q->at = t;

```
while (n1 != n)
                                                                        m = (s1[i].end -
                                                                                                       }
                                                                   s1[i].start);
         t = p->bt;
                                                                        for (k = 0; k < m /
                                                                                                       for (i = 0; i <
         p->bt = q->bt;
                                      p = get_p();
                                                                                                    no_of_frames; ++i)
         q->bt=t;
                                      if (p == NULL)
                                                                   2; k++)
         t = p->ct;
                                                                          printf("-");
                                                                                                       {
                                                                        printf("%s",
                                                                                                         frames[i] = -1;
         p->ct = q->ct;
                                        time++;
                                                                   s1[i].pname);
         q->ct=t;
                                        s[k].start = prev;
                                                                        for (k = 0; k < (m +
         t = p -> bt1;
                                                                                                       for (i = 0; i <
         p->bt1 = q-
                                 strcpy(s[k].pname,
                                                                   1) / 2; k++)
                                 "*");
                                                                          printf("-");
                                                                                                    no_of_pages; ++i)
>bt1;
                                                                        printf("%d",
                                                                                                       {
         q->bt1=t;
                                        s[k].end = time;
         t = p -> p;
                                        prev = time;
                                                                   s1[i].end);
                                                                                                         flag1 = flag2 = 0;
         p->p = q->p;
                                        k++;
                                                                     }
         q->p=t;
                                      }
                                                                   }
                                                                                                         for (j = 0; j <
       }
                                      else
                                                                   int main()
                                                                                                    no_of_frames; ++j)
                                      {
       q = q->next;
    }
                                        time += p->bt1;
                                                                     accept_info();
                                                                                                            if (frames[j] ==
                                        s[k].start = prev;
                                                                     sort();
                                                                                                    pages[i])
    p = p->next;
                                                                     pnp();
                                                                                                            {
  }
                                 strcpy(s[k].pname, p-
                                                                     print_output();
                                                                                                              flag1 = flag2 =
}
                                 >pname);
                                                                     print_gantt_chart();
                                                                                                    1;
int time;
                                        s[k].end = time;
                                                                     return 0;
                                                                                                              break;
NODE *get_p()
                                        prev = time;
                                                                                                            }
                                        k++;
  NODE *p, *min_p =
                                                                                                         if (flag1 == 0)
                                        p->ct = time;
                                                                   slip 17
                                        p->bt1 = 0;
                                                                   1->
NULL;
  int min = 9999;
                                        n1++;
                                                                   #include <stdio.h>
                                                                                                            for (j = 0; j <
  p = first;
                                                                   int main()
                                                                                                     no_of_frames; ++j)
  while (p != NULL)
                                      print_input();
                                      sort();
                                                                     int no_of_frames,
                                                                                                              if (frames[j]
    if (p->at <= time
                                   }
                                                                   no_of_pages,
                                                                                                    == -1)
&& p->bt1 != 0 &&
                                 }
                                                                   frames[10], pages[30],
       p->p < min)
                                 void
                                                                   temp[10], flag1, flag2,
                                                                                                                faults++;
    {
                                 print gantt chart()
                                                                        flag3, i, j, k, pos,
                                                                                                                frames[j] =
       min = p -> p;
                                                                   max, faults = 0;
                                                                                                    pages[i];
       min_p = p;
                                    int i, j, m;
                                                                     printf("Enter
                                                                                                                flag2 = 1;
                                    s1[0] = s[0];
                                                                   number of frames: ");
                                                                                                                break;
    }
                                                                     scanf("%d",
                                                                                                              }
    p = p->next;
                                                                                                           }
                                   for (i = 1, j = 0; i < k;
                                                                   &no_of_frames);
                                                                                                         }
                                 i++)
  return min p;
                                                                     printf("Enter
                                      if
                                                                   number of pages: ");
                                                                                                         if (flag2 == 0)
struct gantt_chart
                                                                     scanf("%d",
{
                                 (strcmp(s[i].pname,
                                 s1[j].pname) == 0)
                                                                   &no_of_pages);
                                                                                                            flag3 = 0;
  int start;
                                        s1[j].end =
  char pname[30];
                                                                     printf("Enter page
  int end;
                                 s[i].end;
                                                                                                            for (j = 0; j <
} s[100], s1[100];
                                                                   reference string: ");
                                      else
                                                                                                     no_of_frames; ++j)
int k;
                                        s1[++j] = s[i];
                                                                                                            {
void pnp()
                                                                     for (i = 0; i <
                                                                                                              temp[j] = -1;
                                    printf("%d",
                                                                   no_of_pages; ++i)
                                                                     {
  int prev = 0,
                                 s1[0].start);
                                                                                                              for (k = i + 1; k)
                                    for (i = 0; i \le j; i++)
                                                                        scanf("%d",
                                                                                                    < no_of_pages; ++k)
    n1 = 0;
  NODE *p;
                                                                   &pages[i]);
                                                                                                              {
```

```
if (frames[j]
                                   }
                                                                                                  }
                                                                        last->next = p;
                                                                                                  void sort()
== pages[k])
                                                                      last = p;
                                   printf("\n\nTotal
                                                                   }
              temp[j] =
                                Page Faults = %d",
                                                                                                    NODE *p, *q;
                                                                 }
k;
                                faults);
                                                                 void print_output()
                                                                                                    int t;
              break;
                                                                                                    char name[20];
           }
                                   return 0;
                                                                   NODE *p;
                                                                                                    p = first;
         }
                                }
                                                                                                    while (p->next !=
                                                                   float avg_tat = 0,
       }
                                                                                                  NULL)
                                                                 avg wt = 0;
                                2->
                                                                                                    {
                                                                 printf("pname\tat\tbt\
       for (j = 0; j <
                                #include <stdio.h>
                                                                                                       q = p->next;
no_of_frames; ++j)
                                #include <stdlib.h>
                                                                 tct\ttat\twt\n");
                                                                                                       while (q != NULL)
                                #include <string.h>
                                                                    p = first;
         if (temp[j] == -
                                                                   while (p != NULL)
                                   typedef struct
                                                                                                         if (p->at > q->at)
                                process_info
1)
                                                                                                           strcpy(name,
                                                                      int tat = p->ct - p-
                                                                                                  p->pname);
           pos = j;
                                   char pname[20];
                                                                 >at;
           flag3 = 1;
                                   int at, bt, ct, bt1;
                                                                      int wt = tat - p-
                                                                                                           strcpy(p-
           break;
                                   struct process_info
                                                                 >bt;
                                                                                                  >pname, q->pname);
         }
                                 *next;
                                                                                                           strcpy(q-
                                                                      avg_tat += tat;
       }
                                } NODE;
                                                                      avg_wt += wt;
                                                                                                  >pname, name);
                                int n;
                                                                                                           t = p->at;
                                NODE *first, *last;
                                                                 printf("%s\t%d\t%d\t%
       if (flag3 == 0)
                                                                                                           p->at = q->at;
                                void accept_info()
                                                                 d\t%d\t%d\n",
                                                                                                           q->at=t;
         max =
                                                                          p->pname, p-
                                   NODE *p;
temp[0];
                                                                 >at, p->bt, p->ct, tat,
                                                                                                           t = p->bt;
         pos = 0;
                                   int i;
                                                                 wt);
                                                                                                           p->bt = q->bt;
                                   printf("Enter no.of
                                                                                                           q->bt=t;
         for (j = 1; j <
                                process:");
                                                                      p = p->next;
                                                                                                           t = p->ct;
no_of_frames; ++j)
                                   scanf("%d", &n);
                                                                                                           p->ct = q->ct;
         {
                                   for (i = 0; i < n; i++)
                                                                    printf("Avg
                                                                                                           q->ct=t;
           if (temp[j] >
                                                                 TAT=%f\tAvg
                                                                                                           t = p->bt1;
                                     p = (NODE)
                                                                 WT=%f\n'',
                                                                                                           p->bt1=q-
max)
                                 *)malloc(sizeof(NODE))
                                                                        avg_tat / n,
                                                                                                  >bt1;
              max =
                                                                 avg_wt / n);
                                                                                                           q->bt1=t;
temp[j];
                                     printf("Enter
                                                                                                         }
                                                                 void print_input()
              pos = j;
                                process name:");
                                                                                                         q = q->next;
           }
                                     scanf("%s", p-
         }
                                >pname);
                                                                    NODE *p;
                                     printf("Enter
                                                                    p = first;
                                                                                                       p = p->next;
                                                                                                    }
       frames[pos] =
                                arrival time:");
                                     scanf("%d", &p-
                                                                                                  }
pages[i];
       faults++;
                                >at);
                                                                 printf("pname\tat\tbt\
                                                                                                  int time;
    }
                                     printf("Enter first
                                                                 n");
                                                                                                  NODE *get_fcfs()
                                CPU burst time:");
                                                                   while (p != NULL)
                                                                                                  {
    printf("\n");
                                     scanf("%d", &p-
                                                                   {
                                                                                                    NODE *p;
                                >bt);
                                                                                                    p = first;
    for (j = 0; j <
                                                                 printf("%s\t%d\t%d\n"
                                                                                                    while (p != NULL)
                                     p->bt1 = p->bt;
no_of_frames; ++j)
                                     p->next = NULL;
                                                                          p->pname, p-
                                                                                                       if (p->at <= time
       printf("%d\t",
                                     if (first == NULL)
                                                                 >at, p->bt1);
                                                                                                  && p->bt1 != 0)
frames[j]);
                                       first = p;
                                                                      p = p->next;
                                                                                                         return p;
                                                                   }
    }
                                     else
                                                                                                       p = p->next;
```

}	for (i = 1, j = 0; i < k;	scanf("%d", &n);	if (time[i] < min)
return NULL;	i++)	printf("Enter no.of	{
}	{	references:");	min = time[i];
struct gantt_chart	if	scanf("%d", &m);	min_i = i;
{	(strcmp(s[i].pname,	printf("Enter	}
int start;	s1[j].pname) == 0)	reference string:\n");	}
char pname[30];	s1[j].end =	for (i = 0; i < m; i++)	return min_i;
int end;	s[i].end;	{	}
} s[100], s1[100];	else	printf("[%d]=", i);	void Iru()
int k;	s1[++j] = s[i];	scanf("%d",	{
void fcfs()	}	&ref[i]);	int i, j, k;
{	printf("%d",	}	for (i = 0; i < m &&
int prev = 0 , $n1 = 0$;	s1[0].start);	}	sp < n; i++)
NODE *p;	for (i = 0; i <= j; i++)	void disp()	{
while (n1 != n)	{	{	k = search(ref[i]);
{	m = (s1[i].end -	int i, j;	if (k == -1)
p = get_fcfs();	s1[i].start);	for (i = 0; i < m; i++)	{
if (p == NULL)	for (k = 0; k < m /	printf("%3d",	frames[sp] =
{	2; k++)	ref[i]);	ref[i];
time++;	, printf("-");	printf("\n\n");	time[sp] = i;
s[k].start = prev;	printf("%s",	for (i = 0; i < n; i++)	faults++;
2 3 2 2 2 P 2 3	s1[i].pname);	{	sp++;
strcpy(s[k].pname,	for (k = 0; k < (m +	for (j = 0; j < m;	for (j = 0; j < n;
"*");	1) / 2; k++)	j++)	j++)
s[k].end = time;	printf("-");	1)	mem[j][i] =
prev = time;	printf("%d",	if (mem[i][j])	frames[j];
k++;	s1[i].end);	printf("%3d",	}
}	}	mem[i][j]);	else
else	}	else	time[k] = i;
Į	int main()	printf(" ");	l (1110-1)
time += p->bt1;	Į)	for (i = 0; i < m; i++)
s[k].start = prev;	accept_info();	printf("\n");	{
S[N].Start = prev,	sort();	}	k = search(ref[i]);
strcpy(s[k].pname, p-	fcfs();	printf("Total Page	if (k == -1)
>pname);	print_output();	Faults: %d\n", faults);	{ !! (!x 1)
s[k].end = time;	print_gantt_chart();	l	sp = get_Iru();
prev = time;	return 0;	int search(int pno)	frames[sp] =
k++;	ı	s	ref[i];
p->ct = time;	ı	ι int i;	time[sp] = i;
p->bt1 = 0;	slip 18	for (i = 0; i < n; i++)	faults++;
p->b(1 - 0, n1++;	1->	(i = 0, i < ii, i++)	for (j = 0; j < n;
11177,	#include <stdio.h></stdio.h>	if (frames[i] ==	
print input():	#define MAX 20		j++) mem[j][i] =
print_input();	int frames[MAX],	pno)	frames[j];
sort();		return i;	rrairies[j],
}	ref[MAX],	roturn 1:) olso
}	mem[MAX][MAX],	return -1;	else
void	faults,	}	time[k] = i;
print_gantt_chart()	sp, m, n, time[MAX];	int get_lru()	}
inti i	void accept()	intimals is said	int main/
int i, j, m;	{ :	int i, min_i, min =	int main()
s1[0] = s[0];	int i;	9999;	{
	printf("Enter no.of	for (i = 0; i < n; i++)	accept();
	frames:");	1	lru();

```
{
  disp();
                                                                    int t;
                                   NODE *p;
  return 0;
                                                                    char name[20];
                                                                                                    int start;
}
                                   float avg tat = 0,
                                                                    p = first;
                                                                                                    char pname[30];
                                avg_wt = 0;
                                                                    while (p->next !=
                                                                                                    int end;
2->
                                                                 NULL)
                                                                                                  } s[100], s1[100];
                                printf("pname\tat\tbt\
                                                                    {
                                                                                                  int k;
#include <stdio.h>
                                tct\ttat\twt\n");
                                                                                                  void fcfs()
                                                                      q = p->next;
                                                                      while (q != NULL)
#include <stdlib.h>
                                   p = first;
                                   while (p != NULL)
#include <string.h>
                                                                                                    int prev = 0, n1 = 0;
                                                                                                    NODE *p;
  struct process_info
                                                                        if (p->at > q->at)
                                     int tat = p->ct - p-
                                                                                                    while (n1 != n)
{
                                                                        {
  char pname[20];
                                >at;
                                                                           strcpy(name,
  int at, bt, ct, bt1;
                                     int wt = tat - p-
                                                                 p->pname);
                                                                                                       p = get_fcfs();
  struct process_info
                                >bt;
                                                                           strcpy(p-
                                                                                                       if (p == NULL)
*next;
                                                                 >pname, q->pname);
} NODE;
                                     avg_tat += tat;
                                                                           strcpy(q-
                                                                                                         time++;
                                     avg_wt += wt;
int n;
                                                                 >pname, name);
                                                                                                         s[k].start = prev;
NODE *first, *last;
                                                                           t = p->at;
void accept_info()
                                printf("%s\t%d\t%d\t%
                                                                           p->at = q->at;
                                                                                                  strcpy(s[k].pname,
                                                                                                  "*");
{
                                d\t%d\t%d\n",
                                                                           q->at=t;
  NODE *p;
                                         p->pname, p-
                                                                                                         s[k].end = time;
  int i;
                                >at, p->bt, p->ct, tat,
                                                                           t = p -> bt;
                                                                                                         prev = time;
  printf("Enter no.of
                                wt);
                                                                           p->bt = q->bt;
                                                                                                         k++;
process:");
                                                                           q->bt=t;
                                                                                                       }
  scanf("%d", &n);
                                                                                                       else
                                     p = p->next;
                                                                           t = p->ct;
  for (i = 0; i < n; i++)
                                   }
                                                                           p->ct = q->ct;
                                   printf("Avg
                                                                                                         time += p->bt1;
                                                                           q->ct=t;
    p = (NODE)
                                TAT=%f\tAvg
                                                                           t = p->bt1;
                                                                                                         s[k].start = prev;
                                WT=%f\n'',
*)malloc(sizeof(NODE))
                                                                           p->bt1=q-
                                       avg_tat / n,
                                                                 >bt1;
                                                                                                  strcpy(s[k].pname, p-
    printf("Enter
                                avg_wt / n);
                                                                           q->bt1=t;
                                                                                                  >pname);
                                                                        }
process name:");
                                                                                                         s[k].end = time;
    scanf("%s", p-
                                void print_input()
                                                                        q = q->next;
                                                                                                         prev = time;
                                                                      }
>pname);
                                                                                                         k++;
    printf("Enter
                                   NODE *p;
                                                                                                         p->ct = time;
arrival time:");
                                   p = first;
                                                                      p = p->next;
                                                                                                         p->bt1 = 0;
    scanf("%d", &p-
                                                                    }
                                                                                                         n1++;
>at);
    printf("Enter first
                                printf("pname\tat\tbt\
                                                                 int time;
                                                                                                       print_input();
CPU burst time:");
                                                                 NODE *get fcfs()
                                n");
                                                                                                       sort();
    scanf("%d", &p-
                                   while (p != NULL)
                                                                                                    }
                                                                                                  }
>bt);
                                                                    NODE *p;
    p->bt1 = p->bt;
                                                                    p = first;
                                                                                                  void
                                printf("%s\t%d\t%d\n"
                                                                    while (p != NULL)
                                                                                                  print_gantt_chart()
    p->next = NULL;
    if (first == NULL)
                                                                      if (p->at <= time
                                         p->pname, p-
                                                                                                    int i, j, m;
       first = p;
                                >at, p->bt1);
                                                                 && p->bt1 != 0)
                                                                                                    s1[0] = s[0];
    else
                                     p = p->next;
                                                                        return p;
       last->next = p;
                                   }
                                                                      p = p->next;
                                                                                                    for (i = 1, j = 0; i < k;
    last = p;
                                }
                                                                    }
                                                                                                  i++)
  }
                                void sort()
                                                                    return NULL;
                                                                                                    {
}
                                   NODE *p, *q;
void print_output()
                                                                 struct gantt_chart
```

if	{	// If the 'n' option was	} job[10];
(strcmp(s[i].pname,	tok[i++]=p;	specified, print the	int n, i, j, tq;
s1[j].pname) == 0)	p=strtok(NULL," ");	count	float avg_tat = 0;
s1[j].end =	}	if (op == 'n')	float avg_wt = 0;
s[i].end;	tok[i]=NULL;	{	// to accept the info
else	}	printf("Number of	about processes
s1[++j] = s[i];	void list(char	entries: %d\n", count);	take_input()
}	*dirname, char op)	}	= · · ·
printf("%d",	{	}	printf("Enter the no
s1[0].start);	DIR *dir;	int main()	of jobs : ");
for (i = 0; i <= j; i++)	struct dirent *entry;	{	scanf("%d", &n);
{	int count = 0;	char	printf("Enter the
m = (s1[i].end -	// Open the directory	buff[80],*args[10];	time Quantum: ");
s1[i].start);	dir =	int pid;	scanf("%d", &tq);
for $(k = 0; k < m /$	opendir(dirname);	while(1)	for (i = 0; i < n; i++)
2; k++)	if (dir == NULL)	{	{
printf("-");	{	printf("myshell\$ ");	printf("Enter the
printf("%s",	printf("Directory %s	fflush(stdin);	name of the job: ");
s1[i].pname);	not found.\n",	fgets(buff,80,stdin);	scanf("%s",
for (k = 0; k < (m +	dirname);	buff[strlen(buff)-	&job[i].name);
1) / 2; k++)	return;	1]='\0';	printf("Enter the
printf("-");	}	make_toks(buff,args);	arrival time of the job :
printf("%d",	// Iterate through the	if (strcmp(args[0],	");
s1[i].end);	entries in the directory	"list") == 0)	scanf("%d",
}	while ((entry =	{	&job[i].at);
}	readdir(dir)) != NULL)	list(args[2],	printf("Enter the
int main()	{	args[1][0]);	burst time of the job:
{	switch (op)	}	");
accept_info();	{	else	scanf("%d",
sort();	case 'f':	{	&job[i].bt);
fcfs();	// Display the	pid = fork();	job[i].tbt =
print_output();	filename	if (pid > 0)	job[i].bt;
print_gantt_chart();	printf("%s\n", entry-	wait();	printf("\n\n");
return 0;	>d_name);	else	}
}	break;	{	}
	case 'n':	if (execvp(args[0],	
slip 19	// Increment the	args) == -1)	// to sort the
1->	count	printf("Bad	processes by arriaval
#include <unistd.h></unistd.h>	count++;	command.\n");	time
#include <sys types.h=""></sys>	break;	}	sort()
#include <sys stat.h=""></sys>	case 'i':	}	{
#include <fcntl.h></fcntl.h>	// Display the	}	struct job temp;
#include <stdio.h></stdio.h>	filename and inode	return 0;	for (i = 0; i < n; i++)
#include <stdlib.h></stdlib.h>	number	}	{
#include <string.h></string.h>	printf("%s: %lu\n",		for $(j = i + 1; j < n;$
#include <dirent.h></dirent.h>	entry->d_name, entry-	2->	j++)
void make_toks(char	>d_ino);	#include <stdio.h></stdio.h>	{
*s, char *tok[])	break;	#include <string.h></string.h>	if (job[i].at >
{	}	struct job	job[j].at)
int i=0;	}	{	{
char *p;	closedir(dir);	char name[20];	temp = job[i];
p = strtok(s," ");		int at, bt, ct, tat, wt,	job[i] = job[j];
while(p!=NULL)		st, tbt;	job[j] = temp;

}	if (jno == n)	}	int n =
}	{	process();	atoi(args[1] + 1);
}	jno = 0;	print_output();	// Open the
}	}	}	file
// to calculate the tat n	if (cnt == n)		FILE *fp =
wt	break;	slip 20	fopen(args[2], "r");
void process()	}	1->	if (fp == NULL)
{	}	#include <sys types.h=""></sys>	{
int jno = 0 , cnt = 0 ,	// to print the output	#include <sys stat.h=""></sys>	
time = job[0].at;	table	#include <fcntl.h></fcntl.h>	printf("Error opening
	void print_output()	#include <stdio.h></stdio.h>	file %s.\n", args[2]);
printf("\n********	{	#include <stdlib.h></stdlib.h>	return 1;
Gantt	printf("\n\n");	#include <unistd.h></unistd.h>	}
chart*******\n");	printf("\n	#include <string.h></string.h>	// Read and
while (1)	");	void make_toks(char	print the first n lines
{	printf("\n pname	*s, char *tok[])	char line[80];
if (job[jno].tbt !=	at bt tat wt");	{	int count = 0;
0)	printf("\n	int i = 0;	while (count <
{	");	char *p;	n && fgets(line, 80, fp)
printf(" %d %s	for (i = 0; i < n; i++)	p = strtok(s, " ");	!= NULL)
", time,	{	while (p != NULL)	{
job[jno].name);	printf("\n%s %d	{	printf("%s",
if (job[jno].tbt	%d %d %d",	tok[i++] = p;	line);
>= tq)	job[i].name, job[i].at,	p = strtok(NULL, "	count++;
(:ab[:ma] +b+ —	job[i].bt, job[i].tat,	");	// Class the
job[jno].tbt =	job[i].wt);	} +al:[:] = NUU .	// Close the
job[jno].tbt - tq; time = time +	avg_tat = avg_tat	tok[i] = NULL;	file
	+ (float)(job[i].tat);	int main()	fclose(fp); າ
tq;	avg_wt = (float)avg_wt +	{	// If the '-a'
else	(float)(job[i].wt);	char buff[80],	option is specified
{ 	\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	*args[10];	else if
time = time +	, printf("\n	int pid;	(strcmp(args[2], "-a")
job[jno].tbt;	");	while (1)	== 0)
job[jno].tbt =	printf("\nThe avg of	{	0)
0;	the turn around time is	printf("myshell\$	// Open the
}	%f", avg_tat / n);	");	file
printf("%d ",	printf("\nThe avg of	fflush(stdin);	FILE *fp =
time);	the waiting time is %f",	fgets(buff, 80,	fopen(args[2], "r");
if (job[jno].tbt	avg_wt / n);	stdin);	if (fp == NULL)
== 0)	}	buff[strlen(buff) -	{
{	main()	1] = '\0';	·
job[jno].ct =	{	make_toks(buff,	printf("Error opening
time;	take_input();	args);	file %s.\n", args[2]);
job[jno].tat =	process();	if (strcmp(args[0],	return 1;
time - job[jno].at;	print_output();	"typeline") == 0)	}
job[jno].wt =	for (i = 0; i < n; i++)	{	// Read and
job[jno].tat -	{	if (args[1][0] ==	print the entire file
job[jno].bt;	job[i].tbt =	'+')	char line[80];
cnt++;	job[i].bt = rand() % 10	{	while
}	+ 1;	// Read the	(fgets(line, 80, fp) !=
}	job[i].at = job[i].ct	line count	NULL)
jno++;	+ 2;		

```
printf("%s",
                                                                                                    NODE *p, *min_p =
                                                                 printf("%s\t%d\t%d\n"
                                                                                                 NULL;
line);
                                     p->bt1 = p->bt;
         // Close the
                                                                                                    int min = 9999;
                                     p->next = NULL;
file
                                                                                                    p = first;
                                     if (first == NULL)
                                                                         p->pname, p-
         fclose(fp);
                                       first = p;
                                                                 >at, p->bt1);
                                                                                                    while (p != NULL)
       }
                                     else
                                                                      p = p->next;
       else
                                       last->next = p;
                                                                   }
                                                                                                      if (p->at <= time
                                                                 }
                                                                                                  && p->bt1 != 0 &&
                                     last = p;
         printf("Invalid
                                  }
                                                                 void sort()
                                                                                                         p->bt1 < min)
option.\n");
                                void print_output()
                                                                   NODE *p, *q;
                                                                                                         min = p->bt1;
       }
    }
                                                                   int t;
                                                                                                         min_p = p;
  }
                                   NODE *p;
                                                                   char name[20];
}
                                  float avg_tat = 0,
                                                                   p = first;
                                                                                                      p = p->next;
                                                                   while (p->next !=
                                avg_wt = 0;
2->
                                                                 NULL)
                                                                                                    return min p;
#include <stdio.h>
                                printf("pname\tat\tbt\
                                                                   {
#include <stdlib.h>
                                tct\ttat\twt\n");
                                                                                                 struct gantt_chart
                                                                      q = p->next;
#include <string.h>
                                   p = first;
                                                                     while (q != NULL)
  typedef struct
                                  while (p != NULL)
                                                                     {
                                                                                                    int start;
process_info
                                                                        if (p->at > q->at)
                                                                                                    char pname[30];
                                                                                                    int end;
                                     int tat = p->ct - p-
  char pname[20];
                                                                          strcpy(name,
                                                                                                 } s[100], s1[100];
                                >at;
  int at, bt, ct, bt1;
                                                                                                 int k;
                                     int wt = tat - p-
                                                                 p->pname);
                                                                                                 void sjfnp()
  struct process info
                                >bt;
                                                                          strcpy(p-
*next;
                                                                 >pname, q->pname);
} NODE;
                                                                          strcpy(q-
                                                                                                    int prev = 0, n1 = 0;
                                     avg_tat += tat;
int n;
                                     avg wt += wt;
                                                                 >pname, name);
                                                                                                    NODE *p;
NODE *first, *last;
                                                                                                    while (n1 != n)
                                                                          t = p -> at;
void accept_info()
                                printf("%s\t%d\t%d\t%
                                                                          p->at = q->at;
{
                                d\t%d\t%d\n",
                                                                          q->at=t;
                                                                                                      p = get_sjf();
  NODE *p;
                                                                                                      if (p == NULL)
                                         p->pname, p-
  int i;
                                >at, p->bt, p->ct, tat,
                                                                          t = p->bt;
  printf("Enter no.of
                                wt);
                                                                          p->bt = q->bt;
                                                                                                         time++;
process:");
                                                                          q - bt = t;
                                                                                                         s[k].start = prev;
  scanf("%d", &n);
                                     p = p->next;
                                                                          t = p->ct;
  for (i = 0; i < n; i++)
                                  }
                                                                          p->ct = q->ct;
                                                                                                 strcpy(s[k].pname,
                                                                                                 "*");
                                   printf("Avg
                                                                          q->ct=t;
    p = (NODE)
                                TAT=%f\tAvg
                                                                          t = p->bt1;
                                                                                                         s[k].end = time;
*)malloc(sizeof(NODE))
                                WT=%f\n'',
                                                                          p->bt1 = q-
                                                                                                         prev = time;
                                       avg_tat / n,
                                                                 >bt1;
                                                                                                         k++;
    printf("Enter
                                                                                                      }
                                avg_wt / n);
                                                                          q->bt1=t;
process name:");
                                                                        }
                                                                                                      else
    scanf("%s", p-
                                void print_input()
                                                                        q = q->next;
                                                                     }
>pname);
                                                                                                         time += p->bt1;
    printf("Enter
                                   NODE *p;
                                                                                                         s[k].start = prev;
arrival time:");
                                   p = first;
                                                                      p = p->next;
    scanf("%d", &p-
                                                                                                 strcpy(s[k].pname, p-
                                                                   }
>at);
                                                                 }
                                                                                                 >pname);
    printf("Enter first
                                printf("pname\tat\tbt\
                                                                 int time;
                                                                                                         s[k].end = time;
CPU burst time:");
                                n");
                                                                 NODE *get_sjf()
                                                                                                         prev = time;
    scanf("%d", &p-
                                  while (p != NULL)
                                                                 {
                                                                                                         k++;
>bt);
                                                                                                         p->ct = time;
```

p->bt1 = 0;	1->	printf("Enter the	strcpy(prev_job,
n1++;	#include <stdio.h></stdio.h>	burst time of the job:	cur_job);
}	#include <sys types.h=""></sys>	");	if (cnt == n)
print_input();	#include <unistd.h></unistd.h>	scanf("%d",	break;
sort();	int main()	&job[i].bt);	}
}	{	printf("Enter the	printf("End Time
}	int pid=fork();	name of the job: ");	%d", time);
void	if(pid>0){	scanf("%s",	}
print_gantt_chart()	printf("I am parent	job[i].name);	•
{	process\n");	printf("Enter the	int getjob(int time)
int i, j, m;	printf("PID =	priority: ");	{
s1[0] = s[0];	%d\n\n",getpid());	scanf("%d",	int i, min = 0, k;
	}	&job[i].p);	for $(i = 0; i < n; i++)$
for (i = 1, j = 0; i < k;	else if(pid==0){	job[i].tbt =	if (job[i].at <= time
i++)	printf("I am child	job[i].bt;	&& job[i].tbt != 0)
{	process\n");	printf("\n\n");	{
if	printf("PID =	}	if (min <
(strcmp(s[i].pname,	%d\n",getpid());	}	job[i].p)
s1[j].pname) == 0)	}	// to calculate the tat n	{
s1[j].end =	else{	wt	min = job[i].p;
s[i].end;	printf("Failed to	void process()	k = i;
else	create child process");	{	}
s1[++j] = s[i];	}	int time = 0, cnt = 0,	}
}	return 0;	i;	return k;
printf("%d",	}	char prev_job[10],	}
s1[0].start);		cur_job[10];	// to print the output
for (i = 0; i <= j; i++)	2->	while (1)	table
{	#include <stdio.h></stdio.h>	{	<pre>void print_output()</pre>
m = (s1[i].end -	#include <string.h></string.h>	jno = getjob(time);	{
s1[i].start);	struct job	job[jno].tbt;	printf("\n\n");
for $(k = 0; k < m /$	{	strcpy(cur_job,	printf("\n
2; k++)	char name[20];	job[jno].name);	");
printf("-");	int at, bt, ct, tat, wt,	if (strcmp(cur_job,	printf("\n pname at
printf("%s",	st, tbt, p;	prev_job) != 0)	bt tat wt ");
s1[i].pname);	} job[10];	printf("%d>	printf("\n
for $(k = 0; k < (m +$	int jno, n, i, j;	%s", time,	");
1) / 2; k++)	float avg_tat = 0;	job[jno].name);	for (i = 0; i < n; i++)
printf("-");	float avg_wt = 0;	time++;	{
printf("%d",		if (job[jno].tbt ==	printf("\n%s %d
s1[i].end);	take_input()	0)	%d %d %d ",
}	{	{	job[i].name, job[i].at,
}	printf("Enter the no	job[jno].ct =	job[i].bt, job[i].tat,
int main()	of jobs : ");	time;	job[i].wt);
{	scanf("%d", &n);	job[jno].tat =	avg_tat = avg_tat
accept_info();	for (i = 0; i < n; i++)	time - job[jno].at;	+ (float)(job[i].tat);
sort();	{	job[jno].wt =	avg_wt =
sjfnp();	printf("Enter the	job[jno].tat -	(float)avg_wt +
print_output();	arrival time of the job:		(float)(job[i].wt);
print_gantt_chart();	");	job[jno].bt;	}
return 0;	scanf("%d",		printf("\n
}	&job[i].at);	cnt++;	");
		}	

printf("\nThe avg of	printf("\nParent	else	p = p->next;
the turn around time is	gets lower priority	last->next = p;	}
%f", avg_tat / n);	%d\n", retnice);	last = p;	}
printf("\nThe avg of	}	}	void sort()
the waiting time is %f",	return 0;	}	{
avg_wt / n);	}	<pre>void print_output()</pre>	NODE *p, *q;
}	Q .2	{	int t;
main()	#include <stdio.h></stdio.h>	NODE *p;	char name[20];
{	#include <stdlib.h></stdlib.h>	float avg_tat = 0,	p = first;
take_input();	#include <string.h></string.h>	avg_wt = 0;	while (p->next !=
process();	typedef struct		NULL)
print_output();	process_info	printf("pname\tat\tbt\	{
for (i = 0; i < n; i++)	{	tp\ttct\ttat\twt\n");	q = p->next;
{	char pname[20];	p = first;	while (q != NULL)
job[i].tbt =	int at, bt, ct, bt1, p;	while (p != NULL)	{
job[i].bt = rand() % 10	struct process_info	{	if (p->at > q->at)
+ 1;	*next;	int tat = p->ct - p-	{
job[i].at = job[i].ct	} NODE;	>at;	strcpy(name,
+ 2;	int n;	int wt = tat - p-	p->pname);
}	NODE *first, *last;	>bt;	strcpy(p-
process();	<pre>void accept_info()</pre>		>pname, q->pname);
print_output();	{	avg_tat += tat;	strcpy(q-
}	NODE *p;	avg_wt += wt;	>pname, name);
	int i;		t = p->at;
	printf("Enter no.of	printf("%s\t%d\t%d\t%	p->at = q->at;
slip 22	process:");	$d\t%d\t%d\t%d\n''$,	q->at = t;
1->	scanf("%d", &n);	p->pname,	
#include <stdio.h></stdio.h>	for (i = 0; i < n; i++)	p->at, p->bt,	t = p->bt;
#include <sys types.h=""></sys>	{	p->p, p->ct, tat, wt);	p->bt = q->bt;
#include <unistd.h></unistd.h>	p = (NODE		q->bt = t;
int main()	*)malloc(sizeof(NODE))	p = p->next;	t = p->ct;
{	;	}	p->ct = q->ct;
int pid = fork();	printf("Enter	printf("Avg	q->ct = t;
int retnice;	process name:");	TAT=%f\tAvg	t = p->bt1;
if (pid == 0)	scanf("%s", p-	WT=%f\n",	p->bt1 = q-
{	>pname);	avg_tat / n,	>bt1;
retnice = nice(-	printf("Enter	avg_wt / n);	q->bt1 = t;
15);	arrival time:");	}	
printf("\nl am	scanf("%d", &p-	void print_input()	t = p->p;
child process, PID =	>at);	{	p->p = q->p;
%d\n", getpid());	printf("Enter first	NODE *p;	q->p = t;
printf("\nChild	CPU burst time:");	p = first;	}
gets higher priority	scanf("%d", &p-		q = q->next;
%d\n", retnice);	>bt);		}
}	printf("Enter	printf("pname\tat\tbt\	n – n > novt.
else if (pid > 0)	priority:");	tp\n");	p = p->next;
{	scanf("%d", &p-	while (p != NULL)	}
retnice = nice(15);	>p);	1	} int time.
printf("\nl am	p->bt1 = p->bt;	nrintf("0/a\+0/d\+0/d\+0/	int time;
parent process, PID =	n Spout - NUUL.	printf("%s\t%d\t%d\t%	NODE *get_p()
%d\n", getpid());	p->next = NULL;	d\n",	\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
	if (first == NULL)	p->pname, p-	NODE *p, *min_p =
	first = p;	>at, p->bt1, p->p);	NULL;

int min = 9999;	}	#include <stdio.h></stdio.h>	printf("Enter
p = first;	print_input();	#include <sys types.h=""></sys>	number of frames: ");
while (p != NULL)	sort();	#include <unistd.h></unistd.h>	scanf("%d",
{	}	int main()	&no_of_frames);
if (p->at <= time	}	{	printf("Enter
&& p->bt1 != 0 &&	void	int pid;	number of pages: ");
p->p < min)	print_gantt_chart()	pid = getpid();	scanf("%d",
{	{	printf("Current	&no_of_pages);
min = p->p;	int i, j, m;	Process ID is : %d\n",	
$min_p = p;$	s1[0] = s[0];	pid);	printf("Enter page
}		printf("\n[Forking	reference string: ");
p = p->next;	for $(i = 1, j = 0; i < k;$	Child Process] \n");	
}	i++)	pid = fork();	for (i = 0; i <
return min_p;	{	if (pid < 0)	no_of_pages; ++i)
}	if	{	{
struct gantt_chart	(strcmp(s[i].pname,	printf("\nProcess	scanf("%d",
{	s1[j].pname) == 0)	can not be created ");	&pages[i]);
int start;	s1[j].end =	}	}
char pname[30];	s[i].end;	else	<i>f</i>
int end;	else	{	for (i = 0; i <
} s[100], s1[100];	s1[++j] = s[i];	if (pid == 0)	no_of_frames; ++i)
int k;	}	{	{
void pnp()	printf("%d",	printf("\nChild	frames[i] = -1;
{ :	s1[0].start);	Process is Sleeping	}
int prev = 0, n1 = 0;	for (i = 0; i <= j; i++)	");	for li = 0. i a
NODE *p;	((a1[:] and	sleep(5);	for (i = 0; i <
while (n1 != n)	m = (s1[i].end -	nrintf(") nOrnhan	no_of_pages; ++i)
1 n = got n():	s1[i].start); for (k = 0; k < m /	printf("\nOrphan	ί flag1 = flag2 = 0:
p = get_p();	2; k++)	Process ID: %d",	flag1 = flag2 = 0;
if (p == NULL)	2, K++) printf("-");	getppid()); }	for (j = 0; j <
{ time++;	printf("%s",	else	no_of_frames; ++j)
s[k].start = prev;	s1[i].pname);	{ /* Parent	110_01_11a111e3, 11j)
S[K].Start - prev,	for (k = 0; k < (m +	Process */	ı if (frames[j] ==
strcpy(s[k].pname,	1) / 2; k++)	printf("\nParent	pages[i])
"*");	printf("-");	Process Completed	{ pages[i])
s[k].end = time;	printf("%d",	");	flag1 = flag2 =
prev = time;	s1[i].end);	}	1;
k++;	}	}	break;
}	}	return 0;	}
else	int main()	}	}
{	{	Q .2	,
time += p->bt1;	accept_info();	#include <stdio.h></stdio.h>	if (flag1 == 0)
s[k].start = prev;	sort();	int	{
	pnp();	main()	for (j = 0; j <
strcpy(s[k].pname, p-	print_output();	{	no_of_frames; ++j)
>pname);	print_gantt_chart();	int no_of_frames,	{
s[k].end = time;	return 0;	no_of_pages,	if (frames[j]
prev = time;	}	frames[10], pages[30],	== -1)
k++;		temp[10], flag1, flag2,	{
p->ct = time;		flag3, i, j,	faults++;
p->bt1 = 0;	slip 23	k, pos, max, faults	frames[j] =
n1++;	1->	= 0;	pages[i];

break;	flag2 = 1;	max =	}	printf("\nparent
	break;	temp[j];	}	process id=%d\n",
Second Content of the content of t	}	pos = j;	}	getppid());
Second Content of the content of t	}	}	void insertionsort(int	system("ps -x");
If (flag2 == 0)	}	}	arr[30], int n)	}
$ \begin{cases} \{ & pages[i]; & for (i=1;i < n; i++) \\ flag3 = 0; & faults++; \\ for (j=0;i < no finitf("\name name name name name name name name $		}	{	else
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	if (flag2 == 0)	frames[pos] =	int i, j, temp;	{
Septical content	{	pages[i];	for (i = 1; i < n; i++)	printf("\nParent
	flag3 = 0;	faults++;	{	process PID = $%d\n''$,
no_of_frames; +j) { {		}	temp = arr[i];	getppid());
temp[i] = -1;	- · · ·		j = i - 1;	·
temp[j] = -1; for (j = 0; j < no_of_frames; ++j) arr[j] > temp) bubblesort(arr, n); printf("\"); for (j = 0; j < n; j++) for (k = i + 1; k) arr[j + 1] = for (j = 0; j < n; j++)	no_of_frames; ++j)	printf("\n");		
for (k = i + 1; k) { arr[j + 1] = printf("\n"); for (i = 0; i < n; i++) for	{		·-	• •
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	temp[j] = -1;	•	arr[j] > temp)	
<pre></pre>		no_of_frames; ++j)	{	
{ frames[j]); if (frames[j]) } } arr[j+] = temp; printf("\n\n\n"); printf("\n\n"); printf("\n\n"); printf("\n\n"); printf("\n\n"); printf("\n\n"); printf(\n"\n\n"); printf(\n"\n"\n"); printf(\n"\n"); printf(\n"\n"\n"); printf(\n"\n"\n"\n"); printf(\n"\n"\n"\n"); printf(\n"\n"\n"\n"\n"\n"\n"\n"\n"\n"\n"\n"\n"\	•	{		• • • •
if (frames[j]	< no_of_pages; ++k)	•	arr[j];	
== pages[k])	{	frames[j]);	j;	
{ temp[j] =		}	}	printf("\n\n\n");
k; break; faults); { fork1() { fork1(); return 0; } } return 0; } n, i, status; } printf("\nEnter the no of values in array	== pages[k])	}	arr[j + 1] = temp;	}
k; break; faults); { fork1() { fork1(); return 0; } } return 0; } n, i, status; } printf("\nEnter the no of values in array	{		}	}
break; faults); { int arr[25], arr1[25], return 0; fork1(); return 0; } return 0; n, i, status; } } printf("\nEnter the no of values in array Q.2 for (j = 0; j <		•	}	int main()
return 0;			void fork1()	{
return 0;	break;	faults);	{ :::t=:::::25] =::::125]	
Printf("\nEnter the no of values in array	}			return 0;
for (j = 0; j < &	}	return 0;		}
for (j = 0; j :"); #include <sys types.h=""> no_of_frames; ++j) slip 24 scanf("%d", &n); #include <sys stat.h=""> { 1-> printf("\nEnter the array elements:"); #include <fcntl.h> if (temp[j] == - #include <sys types.h=""> for (i = 0; i < n; i++)</sys></fcntl.h></sys></sys>	}	}		0.3
no_of_frames; ++j) slip 24 scanf("%d", &n); #include <sys stat.h=""> { 1-> printf("\nEnter the if (temp[j] == - if (temp[j] == - if (temp[j] == - if (temp[j] >= - if (temp</sys>	for (i = 0: i <			
<pre>{ if (temp[j] == -</pre>		clin 24		
<pre>if (temp[j] == - #include <stdio.h> array elements :"); #include <stdio.h> 1)</stdio.h></stdio.h></pre>		•	, , ,	
1) #include <sys types.h=""></sys>		- '	• • •	
<pre>{ #include <unistd.h> scanf("%d", #include <unistd.h> pos = j; #include <stdlib.h> &arr[i]); #include <string.h> flag3 = 1; void bubblesort(int int pid = fork(); void make_toks(char break; arr[30], int n) if (pid == 0) *s, char *tok[]) }</string.h></stdlib.h></unistd.h></unistd.h></pre>			•	
pos = j; #include <stdlib.h> &arr[i]); #include <string.h> flag3 = 1; void bubblesort(int int pid = fork(); void make_toks(char bridge) break; arr[30], int n) if (pid == 0) *s, char *tok[]) } { { { } int i, j, temp; sleep(10); int i = 0; char *p; for (i = 0; i < n; i++)</string.h></stdlib.h>		• • • •		
<pre>flag3 = 1;</pre>				
break; arr[30], int n) if (pid == 0) *s, char *tok[]) } { { } int i, j, temp; for (i = 0; i < n; i++)				
<pre></pre>	•	·		- ·
$eq:formula: for (i = 0; i < n; i++) printf("Child char *p; process PID = %d\n", p = strtok(s, ""); $	}	{	{	{
$eq:formula: for (i = 0; i < n; i++) printf("Child char *p; process PID = %d\n", p = strtok(s, ""); $	}	int i, i, temp;	sleep(10);	int i = 0;
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	•			
	if (flag3 == 0)	{		• •
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	{	for (j = 0; j < n - 1;	getpid());	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	max =	j++)	printf("\nElements	{
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	temp[0];	{	Sorted Using	tok[i++] = p;
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	pos = 0;	if (arr[j] > arr[j +	insertionsort:");	p = strtok(NULL, "
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		1])	insertionsort(arr,	");
	for (j = 1; j <	{	n);	}
if (temp[j] > 1];	no_of_frames; ++j)	temp = arr[j];	printf("\n");	tok[i] = NULL;
	{	arr[j] = arr[j +	for (i = 0; i < n; i++)	}
{ temp; printf("\b"); {	if (temp[j] >	1];	printf("%d,",	void count(char *fn,
	max)	arr[j + 1] =		char op)
} FILE *fh;	{	temp;	printf("\b");	{
		}		FILE *fh;

<pre>int cc = 0, wc = 0, lc = 0; char c; fh = fopen(fn, "r");</pre>	<pre>buff[strlen(buff) - 1] = '\0'; make_toks(buff, args);</pre>	<pre>int binarySearch(int arr[], int low, int high, int key) { while (low <= high) {</pre>	<pre>sprintf(args[1], "%s %d", args[1], arr[i]); }</pre>
if (fh == NULL)	if (strcmp(args[0],	int mid = low + (high -	// Execute the new
{	"count") == 0)	low) / 2;	program with the
printf("File %s not	count(args[2],	if (arr[mid] == key) {	sorted array as
found.\n", fn);	args[1][0]);	return mid;	command line
return;	else	} else if (arr[mid] <	arguments
}	{	key) {	execve(args[0], args,
while ((c = fgetc(fh))	pid = fork();	low = mid + 1;	NULL);
!= EOF)	if (pid > 0)	} else {	} else if (pid > 0) {
{	wait();	high = mid - 1;	// Parent process
if (c == ' ')	else	}	wait(NULL); // Wait
WC++;	{	}	for the child process to
else if (c == '\n')	if	return -1; // Key not	finish
{	(execvp(args[0], args)	found	printf("Child process
wc++;	== -1)	}	completed.\n");
lc++;	printf("Bad		} else {
}	command.\n");	int main() {	// Forking failed
cc++;	}	int arr[] = {5, 2, 8, 12,	printf("Forking
}	}	3};	failed.\n");
fclose(fh);	}	int n = sizeof(arr) /	return 1;
switch (op)	return 0;	sizeof(arr[0]);	}
{	}		
case 'c':		// Fork a child process	return 0;
printf("No.of	slip 25	pid_t pid = fork();	}
characters:%d\n", cc -	1->		
1);	#include <stdio.h></stdio.h>	if (pid == 0) {	2->
break;	#include <stdlib.h></stdlib.h>	// Child process	#include <sys types.h=""></sys>
case 'w':	#include <unistd.h></unistd.h>	char *args[3];	#include <sys stat.h=""></sys>
printf("No.of		args[0] =	#include <fcntl.h></fcntl.h>
words:%d\n", wc);	// Function to sort an	"./binary_search"; //	#include <stdio.h></stdio.h>
break;	integer array using	Name of the new	#include <stdlib.h></stdlib.h>
case 'I':	bubble sort	program to be	#include <unistd.h></unistd.h>
printf("No.of	void bubbleSort(int	executed	#include <string.h></string.h>
lines:%d\n", lc + 1);	arr[], int n) {	args[1] = (char	void make_toks(char
break;	for (int i = 0; i < n - 1;	*)malloc(10 *	*s, char *tok[])
}	i++) {	sizeof(char)); //	{
}	for (int j = 0; j < n - i -	Allocate memory for	int i=0;
int main()	1; j++) {	the sorted array	char *p;
{	if (arr[j] > arr[j + 1]) {	args[2] = NULL; // End	p = strtok(s," ");
char buff[80],	int temp = arr[j];	of arguments	while(p!=NULL)
*args[10];	arr[j] = arr[j + 1];	// Sort the array	\ tok[i++]-n:
int pid; while (1)	arr[j + 1] = temp;	<pre>// Sort the array bubbleSort(arr, n);</pre>	tok[i++]=p; p=strtok(NULL," ");
	S L	bubble301t(a11, 11),	p-strtok(NOLL,),
{ printf("myshell\$,	// Convert the sorted	tok[i]=NULL;
");	}	array to a string	ιοκ[i]-i ν οεε, }
fflush(stdin);	J	sprintf(args[1], "%d",	void search(char *fn,
fgets(buff, 80,	// Function to perform	arr[0]);	char op, char *pattern)
stdin);	binary search on a	for (int i = 1; i < n; i++)	{
· <i>u</i>	sorted integer array	{	FILE *fh;
		•	,

```
int count = 0;
                                 printf("Bad
char line[80];
                                 command.\n");
fh = fopen(fn, "r");
                                 }
if (fh == NULL)
                                 }
                                 }
printf("File %s not
                                 return 0;
found.\n", fn);
return;
while (fgets(line, 80,
fh) != NULL)
if (strstr(line, pattern)
!= NULL)
count++;
if (op == 'a')
printf("%s", line);
}
fclose(fh);
if (op == 'c')
printf("Number of
occurrences: %d\n",
count);
}
int main()
{
char
buff[80],*args[10];
int pid;
while(1)
printf("myshell$ ");
fflush(stdin);
fgets(buff,80,stdin);
buff[strlen(buff)-
1]='\0';
make_toks(buff,args);
if (strcmp(args[0],
"search") == 0)
search(args[2],
args[1][0], args[3]);
else
{
pid = fork();
if(pid>0)
wait();
else
{
if(execvp(args[0],args)
==-1)
```