

Betriebssysteme

Scheduling

Praktikum 5

Fachhochschule Bielefeld
Campus Minden
Studiengang Informatik

Beteiligte Personen:

Name	Matrikelnummer
Peter Dick	1050185

18. Mai 2016

Inhaltsverzeichnis

1	Aufgabe 5.1	3
1.1	Aufgabenstellung	3
1.2	Vorbereitung	3
1.3	Durchführung	3
1.4	Fazit	3
2	Aufgabe 5.2	8
2.1	Aufgabenstellung	8
2.2	Vorbereitung	9
2.3	Durchführung	9
2.4	Fazit	9

Aufgabe 5 - Simulation der Scheduling-Algorithmen

1 Aufgabe 5.1

1.1 Aufgabenstellung

Führen Sie eine Handsimulation der Scheduling-Algorithmen auf dem beiliegenden Übungsblatt mit den angegebenen Werten durch.

1.2 Vorbereitung

Vorlesung drei lesen.

1.3 Durchführung

Aufgaben auf Papier lösen und dann einscannen.

1.4 Fazit

First-Come-First-Served

Die Reihenfolge ist die Eingangsreihenfolge A, B, C, D, E:

FCFS							
	P _{pro}	AT	LT	UT	ST	WT	VT
A	2	0	30	0	0	0	30
B	1	0	20	0	30	30	50
C	5	0	25	0	50	50	75
D	4	0	28	0	45	45	103
E	3	0	78	0	103	103	121

$T_V = \frac{30 + 50 + 75 + 103 + 121}{5} = 75,8$

A	B	C	D	E	
0	30	50	75	103	121

→ 1

Abbildung 1

Shortest Job First

Die Reihenfolge ist E, B, C, D, A:

Shortest Job First						
	Prio	FT	LT	UT	ST	ET
J1	2	0	30	0	91	91
J2	1	0	20	0	18	18
J3	5	0	25	0	38	38
J4	4	0	28	0	63	63
J5	3	0	72	0	0	121

$TV = \frac{121 + 38 + 63 + 91 + 18}{5} = 66,2$

E	A	C	D	F	
0	18	38	63	91	121

→ F

Abbildung 2

Priorisiertes Scheduling

Die Reihenfolge ist C, D, E, A, B:

priorisiertes scheduling							
	Proc	AT	LT	UT	ST	WT	VT
C	1	0	20	8	101	101	101
D	2	0	20	0	101	121	121
E	3	0	25	0	0	25	25
A	4	0	28	0	25	25	53
B	5	0	78	0	53	49	49

$T_V = \frac{101 + 121 + 25 + 53 + 49}{5} = 49,2$

Abbildung 3

Round Robin mit konstanter Zeitscheibe unabhängig von der Priorität

Round Robin mit konstanter Zeitscheibe Zeitscheibe = 10						
A	AT	LT	ST	UT	WT	VT
B	0	30	0	48	48	108
C	0	20	10	40	50	40
D	0	25	20	68	88	115
E	0	28	30	63	93	121
	0	18	40	80	98	

$T_v = \frac{108 + 10 + 115 + 121 + 98}{5} > 102$

~~ABCDEABCDEABCDE~~
gleiche

~~ABCIDIEABICIDIEIACID~~
0 10 20 30 40 50 60 10 20 90 98 108 115 121
→ 1

Abbildung 4

Round Robin mit Zeitscheibendauer proportional zur Priorität

Round Robin mit Zeitscheiben dauer propo zur Prio Teilzeit = 1 Zeitsch. Zeitanteil						
Prozesse					CPU Anteile	Vereinbart
A(2)	B(1)	C(5)	D(4)	E(3)		
20	20	25	28	18		
28	19	20	24	15	75	
26	18	15	20	12	15	
24	14	10	16	9	15	
22	16	5	12	6	15	
20	15	0	8	3	15 $T_{vC} = 25 \cdot 15 = 45$	
18	14	-	4	0	10 $T_{vE} = 15 + 10 = 25$	
16	13	-	0	-	7 $T_{vD} = 85 + 7 = 92$	
0	5	-	-	-	3x8 $T_{vA} = 92 + 3 \cdot 8 = 116$	
0	0	-	-	-	1x5 $T_{vB} = 116 + 5 > 121$	
$T_v = (75 + 85 + 52 + 116 + 121) : 5 = \underline{\underline{94,8}}$						

Abbildung 5

2 Aufgabe 5.2

2.1 Aufgabenstellung

Implementieren Sie folgende in der Vorlesung behandelte Scheduling-Algorithmen:

- First Come First Served
- Shortest Job First
- prioritätsgesteuertes Scheduling
- Round Robin mit konstanter Zeitscheibe unabhängig von der Priorität
- Round Robin mit Zeitscheibendauer proportional zur Priorität

Ihr Programm soll die Algorithmen vollständig simulieren. Außerdem soll die mittlere Verweilzeit bestimmt werden. Verwenden Sie zur Simulation der Abläufe eine Liste von Jobs. Jeder Job speichert seinen Namen, seine Abarbeitungszeit und seine Priorität. Sie können zur vereinfachten Umsetzung die beiliegende Liste verwenden. Die Ausgaben sollen folgendermaßen aussehen und bei jedem Zeitschritt ausgegeben werden:

Beispielausgabe zu First Come First Served:

a wurde abgearbeitet (Aktuelle Zeit: xx).
b wurde abgearbeitet (Aktuelle Zeit: xx).
c wurde abgearbeitet (Aktuelle Zeit: xx).
d wurde abgearbeitet (Aktuelle Zeit: xx).
e wurde abgearbeitet (Aktuelle Zeit: xx).

Mittlere Verweilzeit: xx.xx s

Beispielausgabe zu Round Robin:

Es wird an den Jobs zu folgenden Anteilen gearbeitet:
Es wurde 1s an a gearbeitet.
Es wurde 1s an b gearbeitet.
Es wurde 1s an c gearbeitet.
Es wurde 1s an d gearbeitet.
Es wurde 1s an e gearbeitet.

Es wird an den Jobs zu folgenden Anteilen gearbeitet:

...

Testen Sie alle fünf Algorithmen mit einer Reihe von Jobs verschiedener Größe und vergleichen Sie die Ergebnisse. Testen Sie die Algorithmen ebenfalls für

die im ersten Aufgabenteil vorgegebenen Jobs. Anhand der Ergebnisse können Sie überprüfen, ob Sie die Algorithmen korrekt implementiert haben.

2.2 Vorbereitung

C-Projekt von p04 in Master Mergen.
C-Projekt aufräumen.
Makefile ändern.
Zu branch p05 wechseln.

2.3 Durchführung

Code schreiben und dann testen bzw debuggen.

2.4 Fazit

```
typedef struct Job {
    LIST_NODE_HEADER(struct Job);
    char *name;
    int id;
    int laufzeit;
    int prio;
} JobNode;

typedef struct {
    LIST_HEADER(JobNode);
} JobList;
```

Hier wird der Job und die JobList definiert.

```
int compareJobNodes_id(JobNode *n1, JobNode *n2, void *userData) {
    if (n1->id > n2->id) {
        return 1;
    }
    else if (n1->id < n2->id) {
        return -1;
    }
    else {
        return 0;
    }
}
```

Die Methode vergleicht die Listen Element anhand der Id.

```

int compareJobNodes_lz( JobNode *n1, JobNode *n2, void *userData) {
    if(n1->laufzeit > n2->laufzeit) {
        return 1;
    }
    else if(n1->laufzeit < n2->laufzeit) {
        return -1;
    }
    else {
        return 0;
    }
}

```

Die Methode vergleicht die Listen Element anhand der Laufzeit.

```

int compareJobNodes_pr( JobNode *n1, JobNode *n2, void *userData) {
    if(n1->prio > n2->prio) {
        return -1;
    }
    else if(n1->prio < n2->prio) {
        return 1;
    }
    else {
        return 0;
    }
}

```

Die Methode vergleicht die Listen Element anhand der Priorität.

```

void initJobs(void) {
    List_init(&jobs);
    JobNode *a, *b, *c, *d, *e;
    a = LIST_NEW_NODE(JobNode);
    a->name = "A";
    a->id = 0;
    a->laufzeit = 30;
    a->prio = 2;

    b = LIST_NEW_NODE(JobNode);
    b->name = "B";
    b->id = 1;
    b->laufzeit = 20;
    b->prio = 1;

    c = LIST_NEW_NODE(JobNode);
    c->name = "C";
    c->id = 2;
}

```

```

c->laufzeit = 25;
c->prio = 5;

d = LIST_NEW_NODE( JobNode );
d->name = "D";
d->id = 3;
d->laufzeit = 28;
d->prio = 4;

e = LIST_NEW_NODE( JobNode );
e->name = "E";
e->id = 4;
e->laufzeit = 18;
e->prio = 3;

List_append(&jobs , a );
List_append(&jobs , b );
List_append(&jobs , c );
List_append(&jobs , d );
List_append(&jobs , e );
}

```

Mit der Methode wird eine Beispielliste erstellt.
First Come First Served

```

initJobs ();
JobNode *temp = jobs.head ;
int time = 0;
float vtime = 0;
printf( "First-Come-First-Served\n" );
while(temp != NULL) {
    time += temp->laufzeit ;
    vtime += time ;
    printf( "%s wurde abgearbeitet (Aktuelle Zeit: %d)\n" , temp );
    temp = temp->next ;
}
vtime = vtime / jobs.count ;
printf( "Mittlere Verweilzeit: %f Zeiteinheiten\n\n" , vtime );

```

Die Liste wird Element für Element durch gegangen. Dabei werden die Zeiten aufsummiert(Gesamtzeit und Verweilzeitgesammt) und die Einzelnen Ausgaben getätigt. Am Ende wird die Mittlere Verweilzeit berechnet.
Shortest Job First

```

List_sort(&jobs , (ListNodeCompareFunction) compareJobNodes_lz , NU
temp = jobs.head ;

```

```

time = 0;
vtime = 0;
printf("Shortest-Job-First\n");
while(temp != NULL) {
    time += temp->laufzeit;
    vtime += time;
    printf("%s wurde abgearbeitet (Aktuelle Zeit: %d)\n", temp->name);
    temp = temp->next;
}
vtime = vtime / jobs.count;
printf("Mittlere Verweilzeit: %f Zeiteinheiten\n\n", vtime);

```

Wie bei FCFS nur das die Liste nach Laufzeiten sortiert wird. prioritätsgesteuertes Scheduling

```

List_sort(&jobs, (ListNodeCompareFunction) compareJobNodes_pr, NULL);
temp = jobs.head;
time = 0;
vtime = 0;
printf("prioritätsgesteuertes Scheduling\n");
while(temp != NULL) {
    time += temp->laufzeit;
    vtime += time;
    printf("%s wurde abgearbeitet (Aktuelle Zeit: %d)\n", temp->name);
    temp = temp->next;
}
vtime = vtime / jobs.count;
printf("Mittlere Verweilzeit: %f Zeiteinheiten\n\n", vtime);

```

Wie bei FCFS nur das die Liste nach Prioritäten sortiert wird.
Raund Robin

```

time = 0;
vtime = 0;
int zeitscheibe = 10;
int count = 0;
while(jobs.count != count) {
    temp = jobs.head;
    printf("Es wird an den Jobs zu folgenden Anteilen gearbeitet");
    count = 0;
    while(temp != NULL) {
        if(temp->laufzeit < 1) {
            temp = temp->next;
            count++;
        } else if(temp->laufzeit <= zeitscheibe) {
            printf("Es wurde %d Zeiteinheiten an %s gegeben\n", temp->laufzeit, temp->name);
            temp = temp->next;
            count++;
        }
    }
}

```

```

        time += temp->laufzeit ;
        vtime += time ;
        temp->laufzeit -= zeitscheibe ;
        temp = temp->next ;
    } else {
        printf("Es wurde %d Zeiteinheiten an %s gegeben\n", time, job);
        time += zeitscheibe ;
        temp->laufzeit -= zeitscheibe ;
        temp = temp->next ;
    }
}
}
vtime = vtime / jobs.count ;
printf("Mittlere Verweilzeit: %f Zeiteinheiten\n\n", vtime) ;

```

Zuerst wird nach ID sortiert. Dann werden so oft Zeitscheiben abgezogen bis alle Laufzeiten < 1 sind. Die laufzeiten die < 1 sind werden übersprungen.
Round Robin mit Prioritäten

```

time = 0;
vtime = 0;
count = 0;
while(jobs.count != count) {
    temp = jobs.head;
    printf("Es wird an den Jobs zu folgenden Anteilen gearbeitet\n");
    count = 0;
    while(temp != NULL) {
        if(temp->laufzeit < 1) {
            temp = temp->next;
            count++;
        } else if(temp->laufzeit <= temp->prio) {
            printf("Es wurde %d Zeiteinheiten an %s gegeben\n", time, job);
            time += temp->prio;
            vtime += time;
            temp->laufzeit -= temp->prio;
            temp = temp->next;
        } else {
            printf("Es wurde %d Zeiteinheiten an %s gegeben\n", time, job);
            time += temp->prio;
            temp->laufzeit -= temp->prio;
            temp = temp->next;
        }
    }
}
vtime = vtime / jobs.count ;
printf("Mittlere Verweilzeit: %f Zeiteinheiten\n\n", vtime) ;

```

}

Wie Round Robin nur die Zeinscheiben sind gleich der Priorität.