Betriebssysteme und Softwaretechnik

Abgabe: 18.05.2017

Adrian C. Hinrichs Matr.Nr. 367129
Jeremias Merten Matr.Nr. 367626
Georg C. Dorndorf Matr.Nr. 366511

Aufgabe 3

Aufgabe 3.1

a)

Das Programm kann mittels eines zweiten Terminals und einem entsprechenden `kill`-Befehl gestoppt werden. Als Parameter muss dfür die Process ID übergeben werden. Alternativ kann es im allgemeinen in dem Terminal in dem es gestartet wurde mittels `Ctrl-C` beendet werden.

b)

Die aktuell laufenden Prozesse lassen sich mittels des UNIX-Tools `top` anzeigen. Diese Ausgabe aktualisiert sich ständig und bietet einige weitere Informationen zu den Prozessen. Der Befehl `ps -e` liefert eine Momentaufnahme aller laufenden Prozesse.

c)

Das Programm hängt in einer Endlosschleife, da `i` als `char` initialisiert worden ist und für einen `char` auf den meisten modernen Systemen lediglich acht Bit allokiert werden. Da C die Bit in diesem Fall als signed int betrachtet ist die größte speicherbare Zahl $128 < 500$. Alle 128 Schleifendurchläufe werden die Zahlen also zunächst negativ und nach jedem 256. Durchlauf werden sie aufgrund eines Overflows wieder positiv, nie jedoch größer als 128.

Aufgabe 3.2

a)

Siehe Quellcode.

b)

Das programm `aufgabe32b` lässt sich zur Auswerung der Ausgabe von `forking` über die Shell durch den Befehl

```
./forking | ./aufgabe32b
```

starten. Alle 100000 gelesenen Zeichen wird dann eine Statistik mit relativer Haufigkeit der Zeichen `A`, `B` und `C`, so wie die Anzahl der insgesammt gelesenen Zeichen ausgegeben.

c)

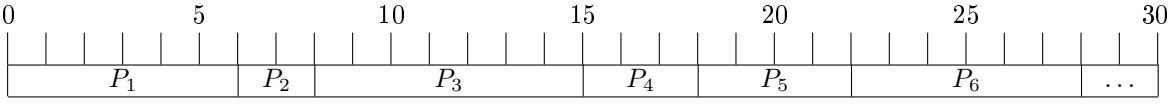
Siehe Quellcode.

Eine Fork-Bomb ist eine Denial of Service Attacke, die den Computer mit sehr vielen Prozessen überlastet. Die `while`-Schleife läuft in unserer Implementation endlos und bei jeder Iteration wird neuer Prozess geforkt.

Moderne Betriebssysteme können den Effekt einer solche Attacke abschwächen wenn sie erkennen, dass ein Prozess unüblich viele forks veranlasst. Diese forks können sie dann einem Elter-Prozess zuordnen und daraufhin die Rechenzeit, die den Prozessen zur verfügung gestellt wir reduzieren. Zum Beispiel indem sie die CPU-Zeit der Gruppe aus Elter und Kindprozessen bündelt und beschränkt. Auch die Anwendung einer non-preemptive mit Priorisierung gesteuerten scheduling Strategie könnte abhilfe schaffen.

Aufgabe 3.3

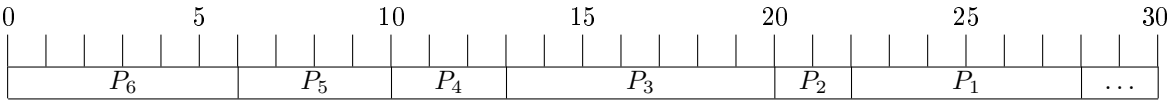
a)



Mittlere Wartezeit

$t_{FCFS} = \frac{6+2+7+3+4}{6} = \frac{22}{6} = 3\ \frac{2}{3}$

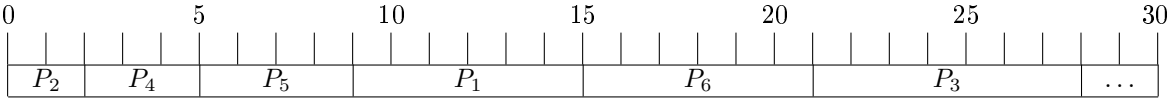
b)



Mittlere Wartezeit

$t_{LCFS} = \frac{6+4+3+7+2}{6} = \frac{22}{6} = 3\ \frac{2}{3}$

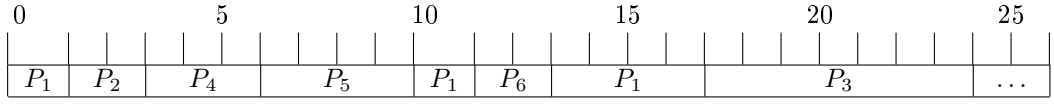
c)



Mittlere Wartezeit

$t_{SJF} = \frac{2+3+4+6+6}{6} = \frac{22}{6} = 3{,}5$

d)



Mittlere Wartezeit

$t_{SJF} = \frac{(17-6-0)+(3-2-1)+(24-7-2)+(6-3-3)+(10-4-4)+(13-2-11)}{6}$

$= \frac{11+0+15+0+2+0}{6} = \frac{28}{6} = 4\ \frac{2}{3}$

Aufgabe 3.4

```c
#include<stdio.h>

int bearbeitungszeiten[] = {6,
                            13,
                            7,
                            3,
                            4,
                            9,
                            10,
                            11};
int prozesse = 8;
int maxQ = 13;

int copy(int a[], int b[], int n)
{
    for(int i = 0; i<=n; i++)
    {
        b[i] = a[i];
    }
    return 0;
}

int main()
{
    printf("#\tQ");
    for(int i=1; i<=prozesse; i++)
    {
        printf("\tP%d",i);
    }
    printf("\tAvg.Time\n");
    printf("#--------");
    for(int i=0; i<=prozesse; i++)
    {
        printf("----------");
    }
    printf("\n");
    int a[prozesse];
    int t;
    int endzeit[prozesse];
    int finished=0;
    int wartezeit=0;
    for(int q = 1; q <= maxQ; q++)
    {
        t=0;
        for(int i=0; i<prozesse; i++)
        {
            endzeit[i]=0;
        }
        copy(bearbeitungszeiten, a, 7);
        finished=0;
        wartezeit=0;
        while(finished<prozesse)
        {
            finished=0;
            for(int i = 0; i< prozesse; i++)
            {
                if(a[i]==0)
                {
                    ++finished;
                }else{
                    if(a[i]>q){
                        a[i]=a[i]-q;
                        t=t+q;
                    }else{
                        t=t+a[i];
                        a[i]=0;
```

```
                        endzeit[i]=t;
                    }
                }
            }
        }
        printf("\t%2d",q);
        for(int i = 0; i<prozesse;i++){
            printf("\t%2d",endzeit[i]);
            wartezeit=wartezeit+endzeit[i]-bearbeitungszeiten[i];
        }
        printf("\t%.2f\n",wartezeit/(float)prozesse);
    }
    return 0;
}
```