Styresystemer og Multiprogrammering

G-opgave #1

Jenny-Margrethe Vej
(rwj935@alumni.ku.dk)
Klaes Bo Rasmussen
(twb822@alumni.ku.dk)

Datalogisk Institut, Københavns Universitet
Blok 3 - 2013

A simple queue using a linked list

Vi har bygget koden op i samme rækkefølge som `queue.h`, og har ydermere lavet en `Makefile`, således man kan køre vores c-fil ved brug af kommandoen `make` i sin terminal.

(a)

Vores længde-funktion returnerer antallet af elementer i en kø. Den starter med længden 0, og er der 1 eller flere elementer, laves der en pointer til den nuværende node i køen, som vi kalder current, og lægger en til. Herefter går vi ned i while-løkken, som bliver ved at tjekke op på, om elementerne i køen er ens, lægger en til længden hver gang, de ikke er ens, og stopper så, når den rammer første element i køen igen.

`enqueue` tilføjer et element til enden af vores kø ved først at allokere pladsen til det vi gerne vil indsætte. Hvis det ikke er muligt at allokere den ønskede hukommelse, printer vi beskeden "Out of memory". I næste if-sætning, kigger vi på, om køen er tom, er den det, indsætter vi et element, som så vil pege på sig selv. Hvis der i forvejen er flere elementer, indsætter vi stadig det ønskede element, men opdaterer samtidig alle pegerne, så de står korrekt efter tilføjelsen af det nye element.

`dequeue` gør brug af samme tankegang som ovenfor, bare "med modsat fortegn" - vi fjerner altså et element her i stedet for at tilføje det, og sørger for også at frigøre pladsen fra det element, vi har fjernet.

For at teste vores løsning, valgte vi at lave en decideret testfil, for at holde den ønskede struktur. Vores tests bygger oven på hinanden, forstået på den måde, at vi først putter noget i vores kø, og derefter tjekker vi længden af elementerne i køen. Dette gør vi et par gange, så vi har mere end et element at teste ud fra. Bagefter tester vi igen længdefunktionen blot sammen med `dequeue`, så vi også ved, at vi kan få den korrekte længde, når vi fjerner elementer. Vi mener, at vores kode fungerer korrekt, da vi ifølge vores tests både kan indsætte, fjerne og beregne længden, jf. opgavetekstens kriterier.

(b)

Grunden til, at `enqueue` og `dequeue` skal have et argument af typen `QNode**` er, at når vi ændrer en værdi af `x`, ved at kalde en procedure, så skal vi bruge en pointer til `x`, som er adressen til `x` i hukommelsen. Kort fortalt er det fordi, at procedurer i C evaluerer efter strategien `call-by-value`.

(c)

Vores sum-funktion er bygget op på samme måde som `length`, med samme måde at løbe gennem køen på. Forskellen består selvfølgelig her i, at sum-funktionen summerer værdien af dataen fra køen. Funktionen er testet sammen med de andre funktioner, da vi alligevel skulle have nogle elementer i

køen at teste ud fra.

`sum` tager 2 argumenter, og det ene satte vi til at være den indbyggede funktion `strlen`. Dog måtte vi lige lave en hjælpefunktion `mystrlen`, da datatypen eksplicit skal være en `char`, og ikke `Data`.

Buenos system calls for basic I/O

(a)

For at besvare denne delopgave, har vi tilføjet de ønskede funktioner til filen `proc/syscall.c`.

Både `read` og `write` er implementeret ud fra simplificeringen i opgaveteksten. Vi gør ikke brug af `fhandle` i denne opgave, da vi kun bruger standardfilerne som via `gcd` håndterer handles automatisk. For at undgå advarsler, når vi kører filerne, starter vi med bare at skrive `fhandle = fhandle;` både til `syscall_read` og `syscall_write`.

Vi har fulgt rådet i opgaveteksten, og brugt samme metoder til håndtering af `gcd`, hvorfor man også vil se, at flere af linjerne indeholder faktisk det samme kode.

Udover de 2 funktioner ovenfor, har vi også tilføjet to cases; `SYSCALL_READ` og `SYSCALL_WRITE`. Her sørger vi for at fange de rigtige cases, og samtidig sørger vi for at putte returværdien i register `v0`, jf. section 6.4 i Buenos Roadmap. Fra vores cases bliver vores 2 funktioner `syscall_read` og `syscall_write` kaldt med de korrekte argumenter fra registrene `a1--a3`.

(b)

Vores testfil ligger i mappen tests.

Når vi giver vores `syscall_write` et antal karakterer at skrive til `stdout`, finder den de første ledige, uden at vente på eventuel input undervejs. Det betyder, at hvis vi skrev 60, ville den finde 60 tegn, udskrive dem og først derefter afvikle næste systemkald. Man kan altså sørge for, at den kun skriver det, man vil have, ved at give den korrekte længde med på den streng, man vil udskrive.

`syscall_read` læser en karakter fra `stdin` og gemmer derefter i en specificeret buffer.

Vores udførte tests giver de forventede output, og vi konkluderer derfor, vi har løst opgaven som ønsket.