Obligatorisk opgave 3

Operativsystemer og C

Bachelor in Software Development,
IT-University of Copenhagen

Jakob Melnyk, jmel@itu.dk
Frederik Lysgaard, frly@itu.dk
Jacob Claudius Grooss, jcgr@itu.dk

November 30 th, 2012

Contents

1 Forord

I denne rapport dokumenterer vi vores valg i forhold til implementationen af opgaverne i Obligatorisk opgave 3.

Kildekode og testdokumentation kan findes i appendix på side 4. Vores git repository kan findes på https://github.com/esfdk/BOSC/tree/master/oo3.

2 Beskrivelse af implementation

2.1 Opgave 1

2.1.1 Del 1

ADD untagger de to øverste elementer på stakken, ligger dem sammen, tagger den nye værdi og ligger den på toppen af stakken.

CSTI I tager den næste værdi i p[ ] arrayet og ligger den på toppen af stakken.

NIL Ligger 0 på toppen af stakken. Hvis der kun ligger rent 0 bits betyder det NIL og ikke integer 0.

IFZERO tager det øverste element af stakken og decrementere stackpointeren med en. Den tjekker om v er en int. Hvis v er en int udtagges v og sammenlignes med nul , ellers sammenlignes v med NIL. Hvis sammenligning er sand i tilfældet med nul bliver program counter sat til den nuværende værdi på (.....) ellers bliver næste instruktion udført.

CONS laver en cons celle ud af de to øverste elementer på stakken, og decrementere så stack pointeren med en.

CAR henter et word fra stakken og tjekker om det er NIL, hvis det ikke er NIL tages det første element af cons cellen og ligges på toppen af stakken i stedet for det hentede word.

SETCAR henter det øverste element på stakken og et word. Den tager wordets første værdi til at være den udhentede værdi på stakken.

2.1.2 Del 2

Length laver to right shifts, hvilket fjerner de to garbage collection bites. Derefter bruger den bitwise AND til at sammenligne length bitsne med 0x003FFFF. Dette giver os værdien af n bitsne.

Color går ind og bruger bitwise AND til at sammenligne farven på cellen med 0011 hvorved den finder cellens faktiske farve.

Paint går ind og ændrer gg til den color der er blevet givet med som argument. eksempelvis laver Paint med arguemnt BLUE gg om til 11.

2.1.3 Del 3

allocate( ) bliver kun kaldt i CONS casen. (umiddelbart ikke andre stedder ind i interpretation loopet)

2.1.4 Del 4

Når der bliver allokeret og der ikke er noget free space.

2.2 Opgave 2, 3 og 4

Igennem opgaverne har vi skulle lave tre forskellige implementationer af garbage collection. En mark-sweep med rekursive funktions kald, en mark-sweep uden rekursive funktions kald og en stop-and-copy med rekursive funktions kald. For at undgå at en fil indeholdte flere implementationer med nogle af dem udkommenteret,

valgte vi at dele dem ud i flere filer. Opgave 2, 3 og 4 findes i listmachineV1.c, opgave 5 i listmachineV2.c og opgave 6 i listmachineV3.c.

Vi har valgt at beskrive vores implementation af opgave 2, 3 og 4 i samme afsnit, da 3 og 4 går ud på at forbedre algoritmen vi implementerede i opgave 2.

Vores implementation of mark-sweep algoritmen består af tre dele; en mark()-funktion, en markPhase()-funktion og en sweepPhase()-funktion.

'mark(word* block)'-funktionen bruges til at farve hvide blokke sorte. Funktionen tjekker først om blokken er farvet (linje 412-416). Er den det, retunerer funktionen, da der ikke skal ske mere. Er den ikke farvet, bliver den farvet sort, hvorefter hvert word der kan nås fra blokken bliver gennemløbet. Hvis wordet ikke er en int og ikke er nil, kaldes mark()-funktionen på wordet (linje 419-428).

'markPhase(int s[], int sp)'-funktionen starter markeringen af blokke. Funktionen looper igennem alle elementer på stacken, og hvis elementet ikke er en int og ikke er nil, castes det til et word og 'mark()' kaldes med wordet some parameter. Disse to funktioner udgør tilsammen mark fasen af algoritmen.

'sweepPhase()'-funktionen er den funktion, som fjerner alt det fundne garbage (linje 445-509). Den looper igennem hele heapen, og tjekker hvilken farve de forskellige words har. Hvis de er grå eller blå, sker der ingenting. Hvis de er sorte bliver de farvet hvide. Hvis de er hvide sker følgende:

1. Det næste word i heapen bliver læst. (linje 459)
2. Der tjekkes om det næste words farve er hvid og om (linje 462)
3. Hvis tjekket bestås, fortæller vi programmet, at der skal afsættes ekstra plads, så vi kan slå blokkene sammen. Derefter indlæses det næste word igen. (linje 464-470)
4. Det tjekkes om der skal afsættes ekstra plads til at slå flere blokke sammen. Hvis der skal, bliver stedet i heapen sat til resultatet af et kald til 'mkheader' macroen med det originale words tag, længden af det originale word plus den ekstra plads og med farven blå. Hvis ikke, bliver stedet i heapen bare farvet blå. (linje 473-482)
5. Til slut bliver wordet lagt over i freelist. (linje 485-487)

Måden vi har implementeret 'sweepPhase()' på, gør at den kan slå flere døde blokke sammen til en død blok, hvorved vi undgår fragmentation.

2.3 Opgave 5

2.4 Opgave 6

A Test

A.1 Opgave 1

A.2 Opgave 2

A.3 Opgave 3

A.4 Opgave 4

A.5 Opgave 5

A.6 Opgave 6

B Kode

B.1 Opgave 1

B.2 Opgave 2

B.3 Opgave 3

B.4 Opgave 4

B.5 Opgave 5

B.6 Opgave 6