

Hermun skautaskála hjá Ísáli

Gunnarr Baldursson & Ragnar Gísli Ólafsson

Apríl 2011

Útdráttur

Ble! Abstract

Efnisyfirlit

1 Inngangur	1
2 Niðurstöður	3
3 Forsendur og Líkan	3
3.1 Umskipting skauta og afköst Skautskála	3
3.2 Bilanir	3
3.3 Einingar og undirkerfi	4
3.4 Vélar og vinnslutímar	5
3.5 Atburðir og kjarnavirkni líkans	5
4 Sannreying Líkans	6
5 Viðauki	6
5.1 Líkanið í forritunarmálinu C	6
5.2 Inntaksgögn líkans	13
5.3 Keyrsluskýrslur	14

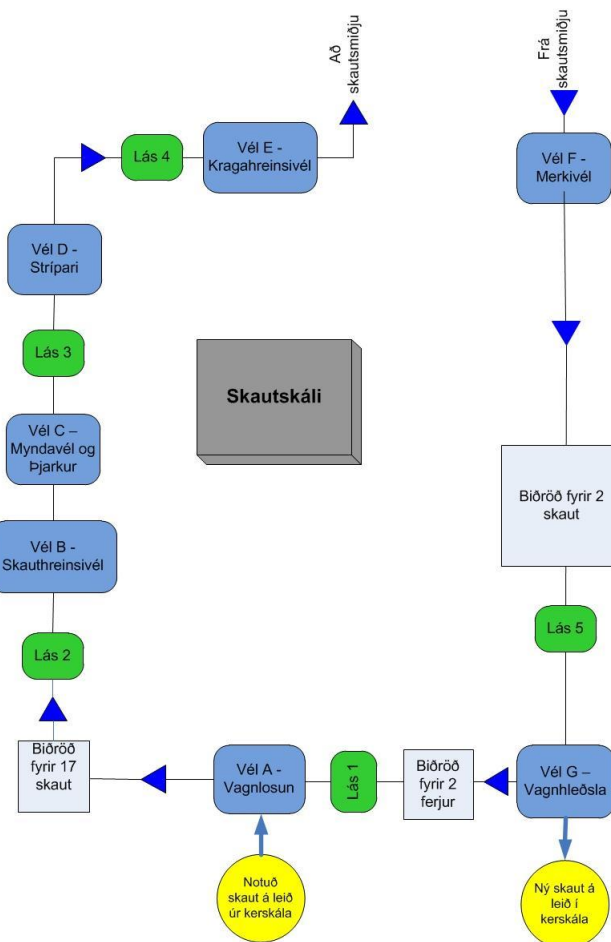
1 Inngangur

Alcan á Íslandi hf., betur þekkt sem Ísál, er hluti af Rio Tinto Alcan, fjölþjóðlegu fyrirtæki sem er stærsti álframleiðandi í heimi. Ísál rekur álverið í Straumsvík sem er ellefta stærsta álverið innan samsteypunnar. Framleiðslugetan er um 185 þúsund tonn og starfsmennirnir eru um 450; vélvirkjar, verkfræðingar, stóriðjugreinar, rafvirkjar, verkafólk, tæknifræðingar, málalarar, skrifstofufólk, bifvéla-virkjar, viðskiptafræðingar, múrarar, matreiðslumenn, rafeindavirkjar, smiðir o.fl.

Ísál rekur þrjá kerskála þar sem að svonefnd skaut eru notuð til rafgreiningar áls. Kerin eru númeruð frá 1001 til 3160 þar sem fyrsta talan stendur fyrir númer skála, og næstu þrjár númer kers í skálanum. Hvert ker hefur 24 skaut sem að slitna með tímanum. Þess vegna þarf að skipta um þau á 26 til 30 daga fresti en það er mismunandi eftir skálum. Skáli 3 hefur stærri skaut og hærri straum, 165 kA og þar endast skaut í 26 daga. Skálar 1 og 2 hafa lægri straum, 133 kA og minni skaut sem að endast í 30 daga. Daglega þarf að skipta út um það bil 404 skautum.

Í kerskála er unnið á þremur vöktum allan sólarhringinn alla daga vikunnar og fer fram skautskipting á hverri vakt. Hver vakt nemur 8 klukkustundum, næturvaktin byrjar á miðnætti, dagvaktin klukkan átta og kvöldvaktin klukkan fjögur. Hver vakt skiptir því um $404/3 = 104$ skaut. Starfsmenn kerskála taka brunnin skaut úr kerum og setja ný skaut í kerin í staðinn. Síðan kemur starfsmaður skautskálans og nær í brunnun skautin sem bíða á vögnum í kerskálanum og flytur þau á sérstakan kæligang. Þar skilur hann þau eftir og nær í staðinn í brunnin skaut sem eru orðin köld og fer með þau í skautskála til hreinsunar. Í hvert skipti sem starfsmaður skautskála sækir brunnin úr kerskála kemur hann með ný skaut. Því er alltaf jafn fjöldi vagna sem fer inn í skautskálann og út úr honum.

Skautin eru flutt á tveimur tengdum vögnum með 12-14 skautum á í einu. Ferðir frá skautskála til kerskála eru aðeins farnar á dagvöktum og kvöldvöktum. Skaut sem þarf að nota á næturna eru því keyrð til kerskála á dag- og kvöldvöktum. Meðaltal fjölda ferða frá skautskála til kerskála eru



Mynd 1: Ferli skautskála

um það bil 30 á sólarhring, eða 15 á vakt. Skautskáli reynir að framleiða þann fjölda skauta sem nemur skautafjölda tveggja vaktu hjá kerskála á hverri vakt, og er því tveimur vöktum á undan.

Fræðileg hámarks afkastageta skautskála eru 52 skaut á klukkustund en vegna bilanna er afkastageta á hverri vakt í besta falli um það bil 40 skaut á klukkustund. Skálinn er framleiðslulína sem að samtímis tekur skautleifar af vögnum og hreinsar ásamt því að taka á móti nýjum skautum og setja á vagnanna. Lestun nýrra skauta og losun brunninna skauta er samtengt ferli, ef ekki er hægt að taka brunninn skaut af vagni þá er heldur ekki hægt að setja ný skaut á vagn.

Framleiðsluferli skautskála hefst þegar skautleifar koma á vögnum að vél A, sem að hífir þær af vögnum. Eftir það fara þær í gegnum vélar B til E þar sem að leifarnar eru hreinsaðar þannig að gaffallinn stendur einn eftir. Gaffallinn heldur síðan áfram inn í aðra byggingu sem að nefnist skautsmiðja, þar sem hann er skoðaður, réttur af og sandblásinn áður en hann fer í steypun þar sem að ný kol eru steyppt við hann. Þá er hann tilbúinn sem nýtt skaut. Þegar þessu ferli er lokið kemur skautið að vél F þar sem það er merkt og sent til vélar G. Vél G lestar skautið á vagn, sem er síðar keyrður til kerskála. Þetta ferli er lýst á Mynd 1.

Skautið er tekið inn í ferlið þannig að það er hengt á ferju sem að er dregin áfram af keðju, sem að fer í gegnum allan skautskálann og inn í skautsmiðjuna og til baka. Ferlið er raðgengt svo ef vél er að afgreiða skaut þarf skautið á eftir að bíða þangað til að vélin hefur lokið sér af. Til að stýra þessu flæði eru svokallaðir lásar staðsettir með regulegu millibili á keðjunni og kúpla þeir ferjum út til að stöðva þær. Þannig geta sum skaut verið á hreyfingu á meðan önnur eru kyrrstæð því að keðjan sjálf stöðvar ekki nema slökkt sé á henni handvirk. Á bak við sumar vélar eru biðraðir en þar bíða skaut eftir afgreiðslu ef að vélin er upptekin. Lása og biðraðir má sjá á Mynd 1. Lásar á undan fullum biðröðum mega ekki sleppa sýnum skautum þangað til að það rúmast til í röðinni. Skaut geta ekki farið framhjá vélum þannig að ef að vél bilar lengi og röð hennar fyllist heldur sá lás sem kemur þar á undan sýnu skauti föstu og þannig koll af kolli. Þannig getur löng bilun stöðvað skautahreinsiferlið

í einhvern tíma þó að keðjan sem ber ferjurnar haldi áfram keyrslu. Hún er þá eins og bílvél með einhvern snúningshraða sem er í hlutlausum gir.

Það er nokkuð slembið hvaða vélar stoppa nema vél F sem að bilar nánast aldrei. Þegar stærri bilanir eiga sér stað þarf að kalla út viðgerðarmenn en í flestum tilfellum tekur það 5 til 30 mínútur að koma bilaðri vél aftur af stað. Skakkt skaut í vél flokkast sem bilun og þá þarf starfsmaður að bakka því út úr vélinni, leiðrétta það og senda inn aftur. Svoleiðis atvik eiga sér stað nokkrum sinnum á sólarhring og er helsta ástæða þess að afköst skautskála nema um það bil 40 skautum á klukkustund. Ef viðgerðartímar eru þeim mun lengri á einhverri vakt þá þarf vaktin sem kemur á eftir að vinna upp framleiðslutapið. Skautskáli keyrir aðeins á dagvöktum og kvöldvöktum.

Framkvæmdir eru hafnar við að auka strauminn í kerum 1 og 2 sem veldur dræmri endingartíma skauta, og koma þau þá til með að endast í 26 til 28 daga eftir straumhækkun. Gerð verður sú nálgun að alltaf sé nóg til af nýjum skautum í skautsmiðju sem koma að vél F. Verkefnið er að herma ferli skautskála með eftirfarandi vangaveltur í huga:

1. Hversu mikið af töfum (í mínútum talið) þolir skautskálinn til að ná lágmarksafköstum?
2. Er það ráðlegt að stækka biðraðir eða bæta við biðröðum?
3. Hve miklu munar það fyrir ferlið ef að starfsmenn koma vélum af stað eins fljótt og þeir geta?
4. Ef tafir eru litlar, hvenær hefur vakt náð lágmarksafköstum?
5. Hversu fljótur er skálinn að vinna upp langar viðgerðatafir?
6. Hvaða áhrif hefur hækkun straums á ferlið?

2 Niðurstöður

3 Forsendur og Líkan

Til að komast að niðurstöðum smíðuðum við líkan sem að hermir eftir ferli skautskála. Í næstu undirgreinum er forsendum líkansins lýst.

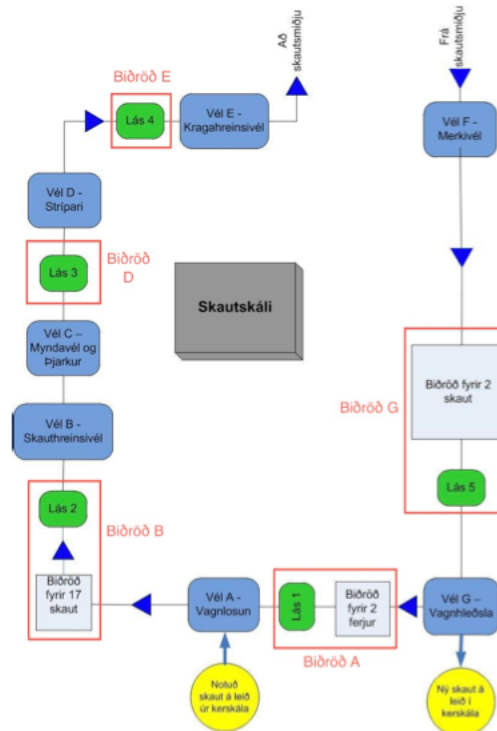
3.1 Umskipting skauta og afköst Skautskála

- Fyrir straumhækkun þá þarf að skipta um skaut í skálum 1 og 2 á 30 daga fresti, og í skála 3 á 26 daga fresti. Það gerir $\frac{24 \cdot 160}{26} + \frac{24 \cdot 160}{30} + \frac{24 \cdot 160}{30} = 403.69$ skaut á dag, þar sem að allir skálar hafa 24 skaut í í hverju kerri og 160 ker eru í hverjum skála. Nefnararnir í formúlunni eru endingadagar skauta í viðeigandi kerskála. Sú tala er námunduð upp í 404 skaut á dag og er það lágmarksafköst skautskála.
- Eftir straumhækkun þarf að skipta um skaut í öllum skálum á 26 daga fresti. Það gerir $\frac{24 \cdot 160}{26} + \frac{24 \cdot 160}{26} = 443.07$ skaut á dag. Sú tala er námunduð upp í 444 skaut á dag og er það lágmarks afkastageta skautskála eftir straumhækkun.
- Fræðileg hámarksafköst skála eru 52 skaut á klst, eða $16 \cdot 62 = 832$ skaut á sólarhring (Skautskálinn starfar aðeins í tvær vaktir, eða 16 klukkustundir á sólarhring).
- Raunveruleg hámarksafköst skála eru 40 skaut á klst, eða $16 \cdot 40 = 640$ skaut á sólarhring (Skautskálinn starfar aðeins í tvær vaktir, eða 16 klukkustundir á sólarhring).

3.2 Bilanir

Samkvæmt verkefnislýsingunni [1] er það nokkuð slembið hvaða vélar bila, að vél F undanskilinni, og að bilanir eiga sér stað nokkrum sinnum á vakt. Gildið á tölunni *nokkrum sinnum* er illa skilgreint en höfundar sammæltust um töluna 8. Fyrir flestar bilanir er viðgerðartíminn 5 til 30 mínútur. Í einhverjum tilfellum þarf að ræsa út viðgerðarmann ef um stórar bilanir er að ræða og slíkar bilanir geta varað í nokkrar klukkustundir. Engin önnur gögn liggja fyrir um bilanir eða tíðni þeirra og þar sem að gögnin eru ekki nákvæmari voru eftirfarandi forsendur gefnar:

- Allir viðgerðartímar liggja á bilinu 5 til 30 mínútur.
- Viðgerðartímar eru veldisreyfðir þannig að mestar líkur eru á viðgerð taki 5 mínútur og minnstar líkur eru á 30 mínútuna viðgerð. Þar sem að bilanir og tafir vegna skakkra skauta í vélum má flokka undir sama hatt þykir höfundum líklegast að um slíkar tafir sé að ræða frekar en vélræna bilun.



Mynd 2: Einingar líkans

- Tímasetningar bilana á sólarhring eru uniform dreifðar.
- Ef að vél A eða G bila eru engar ferðir farnar frá Skautskála til Kerskálanna meðan á viðgerð stendur.

3.3 Einingar og undirkerfi

Þættir skautskála eru dregnar saman í undirkerfi eins og sjá má á eftirfarandi töflu:

Eining	Þættir	Hlutverk
A	Vél A, 14 skauta biðröð í formi vagna	Vagnlosun
B	Vél B, biðröð og lás sem geyma 17 skaut	Skauthreinsivél
C	Vél C	Myndavél og Þjarkur
D	Vél D, einn lás	Strípari
E	Vél E, einn lás	Kragahreinsivél
F	Vél F, einn lás	Merkivél
G	Vél G, biðröð fyrir 2 skaut	Vagnhleðsla

Þessu er lýst á Mynd 2. Vélar B og C geta unnið tvö skaut í einu. [1]

Þegar Mynd 2 er skoðuð má sjá að hægt er að skipta Skautskála upp í tvo helminga sem hefur hvor sitt inntak og sitt úttak. Inntak í vinstri helming kemur frá vél A, og úttak hans fer frá vél E. Inntak í hægri helming kemur frá vél F, en sú nálgun er gerð að þar sé ávallt nóg af nýjum skautum að taka, og úttak þess helmings er vél G, sem að hleður nýju skautunum á vagna. Við heimsókn í Ísal [3] kom fram að bilanir og tafir megi sjaldnast rekja til hægri helmingsins. Þar eru aðeins tvær vélar meðan vinstri hliðin hefur fimm vélar sem að vinna flóknari verk. Af þeim ástæðum er vél F undanskilin hermun. Vél G getur bilað, og ef það gerist stöðvar lestun og losun skauta um þann tíma sem það tekur að gera við bilunina.

Gert er ráð fyrir því að vagnar sem koma með skautaleifar séu ávallt fullskipaðir. Ef það koma tveir fullskipaðir vagnar, með samtals fjórtán skautum, á hálf tíma fresti inn í líkanið við A, og allar vélar hafa vinnutíma sem að er fasti, þá er úttakið við E einnig fasti.

3.4 Vélar og vinnslutímar

Vinnslutími vélar er sá tími sem líður milli þess að skaut kemur að lausri vél og fer frá vélinni aftur. Færslutími er sá tími sem líður milli þess að skaut fer frá vél og kemur að næstu vél. Þeir eru reiknaðir út samkvæmt gagnaskjali, [2]. Þar sem að vélar B og C geta unnið tvö skaut í einu er vinnslutími þeirra helmingaður.

Vél	Vinnslutími	Færslutími
A	70	129.83
B	21.798	122.83
C	12.7	18.98
D	67.41	22.74
E	69.75	0

3.5 Atburðir og kjarnavirkni líkans

Líkanið er atburðadrifið: einhver atburður á sér stað sem að getur skrásett annan atburð og þannig koll af kolli þangað til að keyrslu er lokið. Kjarni líkansins er atburðavinnslan sjálf, hvernig það bregðast skal við þeim atburðum sem að skilgreindir eru. Eftirfarandi atburði skal skilgreina:

- Vagn kemur með brunnin skaut að vél A
- Skaut kemur að vél
- Skaut fer frá vél
- Vél bilar
- Vél löguð
- Endir upphitunartíma
- Endir hermunar

Næstu undirgreinar útskýra hvernig bregðast þarf við þessum viðburðum.

Vagn kemur með brunnin skaut

Vagnar sem koma með hrein skaut geta flutt 12 til 14 skaut saman lagt og er það uniform dreifð slembitala. Fyrir hvert skaut þarf að framkalla atburðinn *skaut kemur að vél*, þar sem að vélin er vél A. Hver slíkur atburður þarf að innihalda eftirfarandi gögn:

- Tímasetningin þegar atburðurinn á sér stað
- Staðsetning skauts í ferlinu
- Tímasetningin þegar skautið kemur fyrst í kerfið
- Raðnúmer skauts

Loks þarf að skrásetja annan *vagn kemur með brunnin skaut* atburð. Þar sem að vagnar koma á um það bil 32 mínútna fresti að meðaltali á dag, meðan unnið er í Skautskála [1], er eðlilegt að koma þeirra sé uniform slembitala milli 28 og 30. Ef að vélar A eða G eru bilaðar þarf að fresta komu næsta vagns um þann tíma sem það tekur við að gera við vélarnar af því að vagnlosunin og lestun eru samtengd ferli [1].

Skaut kemur að vél

Þegar skaut kemur að vél þarf að huga að ýmsu.

- **Er vélin upptekin?**
Ef að vélin er laus skal merkja að skautið hafi fengið þjónustu umsvifalaust. Svo skal skrásetja *Skaut fer frá vél* atburð sem að inniheldur tímasetningu brottfarar og vél sem að farið er frá. Annars skal vista komutíma skauts og setja það í röð þeirrar einingar sem skautið kemur að, ef einhver er.
- **Hefur vélin röð og ef svo er, er röðin full?**
Ef að vélin hefur enga röð eða röðin er full þarf að fresta komu þessa skauts, lásinn sem að heldur því má í rauninni ekki sleppa því þangað til að það rúmast til í röðinni.
- **Er vélin biluð?**
Ef að vélin er biluð þarf að fresta þessum atburði um þann tíma sem að samsvarar viðgerðar-tímanum.

4 Sannreyning Líkans

Heimildir

- [1] Starfsmaður Ísal, *HermunIsal_2011_r2.pdf*. 2011.
- [2] Starfsmaður Ísal, *Millitimar.xls*. 2011.
- [3] *Heimsókn til Ísal*. 21. mars 2011.

5 Viðauki

5.1 Líkanið í forritunarmálinu C

```
1  /*
2  *   isal.c
3  *
4  *
5  *   Created by Gunnarr Baldursson & Ragnar Gisli Olafsson on 4/18/11.
6  *   Copyright 2011 Haskoli Islands. All rights reserved.
7  *
8  */
9
10 #include <stdio.h>
11 #include <string.h>
12 #include <stdlib.h>
13 #include <math.h>
14 #include "simlib/rndlib.h"
15 #include "simlib/simlib.h"
16
17 // EVENTS
18 #define EVENT_WAGEN_UNLOAD_ARRIVAL 1
19 #define EVENT_WAGEN_UNLOAD_DEPARTURE 2
20 #define EVENT_SKAUT_ARRIVAL 3
21 #define EVENT_SKAUT_DEPARTURE 4
22 #define EVENT_MACHINE_FAILURE 5
23 #define EVENT_MACHINE_FIXED 6
24 #define EVENT_END_SIMULATION 7
25 #define EVENT_END_WARMUP 8
26
27 // STREAMS
28 #define STREAM_WAGEN_ARRIVAL 1
29
30 // Other constants
31 #define NUM_MACHINES 7
32 #define WAGEN_LOAD 14
33 #define MACHINES_ON_THE_LEFT_SIDE 5
34 #define MACHINES_ON_THE_RIGHT_SIDE 2
35 #define OPTIMAL_THROUGHPUT 52
36 #define ACTUAL_THROUGHPUT 40
37 #define TRANSFER_ARRAY_LENGTH 11
38 #define PREP_TIME 20.0
39
40 typedef struct
41 {
42     float failtime;
43     float downtime;
44     int machine_nr;
45 } breakdown;
46
47
48 // #define LOADING_TIME_PER_SKAUT
49
50 // Global variables
51 int number_of_machines, min_productivity, min_no_failures, max_no_failures,
    skaut_throughput;
52 float mean_wagen_arrival, std_wagen_arrival, mean_failures, std_failures,
    min_machine_repair_time, max_machine_repair_time, end_warmup_time,
    end_simulation_time;
```

```

53
54
55 int sampst_delays, throughput_time; // variable for queue delays and throughput
    time
56
57 int skaut_id, stream, failure_nr;
58 int queue_size[NUM_MACHINES +1];
59 float machine_broken[NUM_MACHINES +1];
60 breakdown *fail_list;
61
62 int is_machine_busy[NUM_MACHINES +1],
63     queue_size[NUM_MACHINES +1];
64
65 float work_time[NUM_MACHINES + 1],
66     transfer_time[NUM_MACHINES +1]; // +1 is the less preferable simlib
    indexing scheme
67
68
69 float temp_transfer[TRANSFER_ARRAY_LENGTH];
70
71 FILE *infile, *outfile;
72
73 /* Function signatures */
74
75 // Usage: create_machine_fail_events(number_of_failures)
76 // Pre:  init_twister must be called for random number generation
77 // Post:  scheduled events have been created for machines
78 void create_machine_fail_events(int);
79
80
81 // Usage: push_array();
82 // Pre:  we expect that correct values are in transfer array
83 // Post:  our temp_transfer array now has the values in transfer_array
84 void push_array();
85
86 // Usage: pop_array();
87 // Pre:  we expect that correct values are in transfer_temp array
88 // Post:  our transfer array now has the values in transfer_temp
89 void pop_array();
90
91 // Usage: wagen_arrival();
92 // Pre:  EVENT_WAGEN_UNLOAD_ARRIVAL is the next event to be processed
93 // Post:  14 EVENT_SKAUT_ARRIVAL events are next to be processed on the event
    list.
94 void wagen_unload_arrival();
95
96 // Usage: skaut_arrival();
97 // Pre:  EVENT_SKAUT_ARRIVAL is the next event to be processed
98 // Post:  a skaut has been processed by a machine or put in it's queue.
99 //      subsequent events may have been scheduled
100 void skaut_arrival();
101
102 // Usage: skaut_departure();
103 // Pre:  EVENT_SKAUT_DEPARTURE is the next event to be processed
104 // Post:
105 void skaut_departure(); // do we need an event for departure?
106
107 // Usage: machine_failure();
108 // Pre:  EVENT_MACHINE_FAILURE is the next event to be processed
109 // Post:
110 void machine_failure();
111
112 // Usage: machine_fixed();
113 // Pre:  EVENT_MACHINE_FIXED is the next event to be processed
114 // Post:
115 void machine_fixed();
116
117 // Usage: end_warmup();
118 // Post:  SIMLIB statistical variables have been cleared
119 void end_warmup();

```

```

120
121 // Usage: parse_input(input_filename_data,input_filename_time);
122 // Pre:   input_filename_data,input_filename_time of type char[],
123 //        global variables from the input file exist.
124 // Post:  the global variables were assigned values from input_filename,
125 //
126 void parse_input(char[] ,char[]);
127
128 // Usage: x = N(muy, sigma, stream);
129 // Pre:   muy and sigma are of type float
130 //        stream is of type int
131 // Post:  x is a random gaussian distributed variable of type float
132 //        with mean muy and std sigma
133 float N(float muy, float sigma, int stream);
134
135 // Usage: report("the_report.out");
136 // Pre:   the values to be reported have values
137 // Post:  a report on program values and simlib statistics
138 //        have been APPENDED to "the_report.out"
139 void report();
140
141 // Usage: schedule_failures(i);
142 // Pre:   the global variable end_simulation_time has a value, i is of type int
143 // Post:  i failures have been scheduled uniformly on machines
144 //        with ?random? repair times on the interval [min_machine_repair_time,...
145 //        max_machine_repair_time]
146 //        uniformly distributed over the interval 0...end_simulation_time
147 void schedule_failures(int i);
148
149 void queue_is_full();
150
151 int main()
152 {
153 // load datafiles
154     parse_input("adal_inntak.in","velar_og_bidradir.in");
155
156     // initialize arrays and variables
157     if((fail_list = malloc(sizeof(breakdown)))==NULL) {
158         printf("Allocation Error\n");
159         exit(1);
160     }
161
162
163     int b;
164     int stream = 31415;
165     /* for (b=1; b <= number_of_machines; b++) {
166         printf("transfer_time[%d] = %f\n", b,transfer_time[b] );
167         printf("busy %d broken %f \n",is_machine_busy[b],machine_broken[b]);
168     }*/
169     // We perform simulation for "a few" failures per day
170
171     for (failure_nr = min_no_failures; failure_nr < max_no_failures; failure_nr
172         ++ ) {
173
174         memset( is_machine_busy,0, NUM_MACHINES +1 );
175         memset( machine_broken,0, NUM_MACHINES +1);
176         skaut_throughput = 0;
177         sampst_delays = number_of_machines +1;
178         throughput_time = number_of_machines +2;
179
180         skaut_id = 1;
181         skaut_throughput = 0;
182         stream+=3;
183
184         // Initialize rndlib
185         init_twister();
186
187         // Initialize simlib

```



```

188 init_simlib();
189
190 maxatr = 6; // how many attributes do we need?
191
192 /* Schedule machine breakdown time */
193 create_machine_fail_events(failure_nr);
194
195 /* Schedule first wagen arrival */
196 //transfer[3] = 1.0;
197 event_schedule( 10.0, EVENT_WAGEN_UNLOAD_ARRIVAL );
198
199 /* Schedule end of warmup time */
200 event_schedule( end_warmup_time, EVENT_END_WARMUP );
201
202 /* Schedule simulation termination */
203 event_schedule( end_simulation_time, EVENT_END_SIMULATION );
204
205 next_event_type = 0;
206
207
208
209 while (next_event_type != EVENT_END_SIMULATION) {
210
211     timing();
212     /*          printf("event_type = %d, transfer[3] = %f\n",
213         next_event_type, transfer[3]);
214         int k;
215         for (k = 1; k <= number_of_machines; k++)
216             printf("Items in machines/queues %d: %d, %d\n", k, list_size[k],
217                 list_size[number_of_machines + k]);
218         printf("\n");
219     */
220
221     switch (next_event_type) {
222     case EVENT_WAGEN_UNLOAD_ARRIVAL:
223         wagen_unload_arrival();
224         break;
225     case EVENT_SKAUT_ARRIVAL:
226         skaut_arrival();
227         break;
228     case EVENT_SKAUT_DEPARTURE:
229         skaut_departure();
230         break;
231     case EVENT_MACHINE_FAILURE:
232         machine_failure();
233         break;
234     case EVENT_MACHINE_FIXED:
235         machine_fixed();
236         break;
237     case EVENT_END_WARMUP:
238         end_warmup();
239         break;
240     case EVENT_END_SIMULATION:
241         report();
242         break;
243     }
244 }
245 }
246
247 void wagen_unload_arrival()
248 {
249
250     int i;
251     int current_unit = 0;
252     float wagen_arrival_zeit = unirand((mean_wagen_arrival-std_wagen_arrival)
253         *60.0, (mean_wagen_arrival+std_wagen_arrival)*60.0, stream);

```

```

254     for (i = 1; i < NUM_MACHINES + 1; i++) { //delay unload of skaut by the time
        it takes to repair
255     if (machine_broken[i] > 0.0) {
256         event_schedule(sim_time + machine_broken[i], EVENT_WAGEN_UNLOAD_ARRIVAL);
257         return;
258     }
259 }
260
261     if (list_size[number_of_machines + 1] != 0) { // ef allt er enn fullt Áð
        koma með næsta vagn eftir uð b hálftíðma
262     event_schedule(sim_time + wagen_arrival_zeit, EVENT_WAGEN_UNLOAD_ARRIVAL);
263     return;
264 }
265
266     int vagn_magn = WAGEN_LOAD - ((int)unirand(0.0, 3.0, stream)); //12 - 14
        skaut Á hverjum vagni
267     for (i = 1; i <= vagn_magn; i++) {
268
269         transfer[3] = 1.0;
270         transfer[4] = sim_time + (i * 0.01); // skaut entering system time
271         transfer[6] = (float) skaut_id++;
272         //printf("tr4 in wagen: %f\n", transfer[4]);
273         event_schedule(sim_time + (i * 0.01), EVENT_SKAUT_ARRIVAL);
274     }
275
276     event_schedule(sim_time + wagen_arrival_zeit, EVENT_WAGEN_UNLOAD_ARRIVAL);
277 }
278
279
280 void skaut_arrival()
281 {
282     push_array();
283     int current_unit = (int)transfer[3];
284     int i;
285
286     for (i = NUM_MACHINES; i >= current_unit; i--) { //add delay if there is a
        broken machine before current one
287     if (machine_broken[i] > 0.0) {
288         if ((list_size[1 + number_of_machines + current_unit] < queue_size[1 +
            current_unit]) || queue_size[1 + current_unit] == 0) { // if current
            machine is broken then delay it.x
289         event_schedule(sim_time + machine_broken[i] + work_time[current_unit],
            EVENT_SKAUT_ARRIVAL); //also if next queue is full then delay it.
290         return;
291     }
292 }
293 }
294
295 // check if machine is not busy
296 if (list_size[current_unit] == 0 && machine_broken[current_unit] == 0.0) {
297     sampst(0.0, sampst_delays);
298     sampst(0.0, current_unit);
299
300     list_file(FIRST, current_unit); // last := first here because there are only
        to be 0 or 1 items in machine
301
302 // schedule departure after machine processing time
303 pop_array();
304 event_schedule(PREP_TIME + sim_time + work_time[current_unit],
        EVENT_SKAUT_DEPARTURE);
305 } else {
306
307     if (list_size[number_of_machines + current_unit] == queue_size[current_unit])
        {
308
309         event_schedule(PREP_TIME + sim_time + work_time[current_unit],
            EVENT_SKAUT_ARRIVAL); //also if queue is full then delay it.
310
311     } else {
312         transfer[5] = sim_time;

```

```

313     list_file(LAST, number_of_machines + current_unit);
314     //printf("puting skaut in queue: %d\n", current_unit);
315 }
316
317 }
318
319 }
320
321 void skaut_departure()
322 {
323     push_array();
324     int current_unit = (int) transfer[3];
325     int i = 0;
326     for (i = NUM_MACHINES; i >= current_unit; i--) { //add delay if machine is
        broken or there is a broken machine before current one
327     if (machine_broken[i] > 0.0) {
328         if ((i == current_unit) || (list_size[1+number_of_machines +
            current_unit] < queue_size[1+current_unit])) { // if current machine
            is broken then delay it.
329         event_schedule(sim_time + machine_broken[i], EVENT_SKAUT_DEPARTURE); //also
            if next queue is full then delay it.
330         return;
331     }
332     // printf("Size of next queue %d, limit of next queue %d\n", list_size[1+
        number_of_machines + current_unit], queue_size[1+current_unit]);
333     break;
334 }
335 }
336
337 if (current_unit == MACHINES_ON_THE_LEFT_SIDE) {
338     skaut_throughput += 2;
339     sampst(sim_time - transfer[4], throughput_time);
340     list_remove(FIRST, current_unit);
341 } else {
342     list_remove(FIRST, current_unit);
343     pop_array();
344     transfer[3]++;
345     event_schedule(PREP_TIME + sim_time + transfer_time[(int)(transfer[3]) - 1],
        EVENT_SKAUT_ARRIVAL);
346 }
347
348
349 if (list_size[number_of_machines + current_unit] != 0) {
350     pop_array();
351
352     list_file(FIRST, current_unit); // first equals last because size should only
        be 1
353     pop_array();
354
355     list_remove(FIRST, number_of_machines + current_unit);
356     pop_array();
357
358     sampst(sim_time - transfer[5], sampst_delays);
359     sampst(sim_time - transfer[5], current_unit);
360     event_schedule(PREP_TIME + sim_time + work_time[current_unit],
        EVENT_SKAUT_DEPARTURE);
361 }
362 }
363
364
365 void parse_input(char inputfile_data[], char inputfile_time[])
366 {
367
368
369     if ((infile = fopen(inputfile_data, "r")) == NULL) {
370         printf("Could not open file %s\n", inputfile_data);
371     }
372
373     fscanf(infile, "%d %d %d %d %f %f %f %f %f %f", &number_of_machines, &
        min_productivity, &min_no_failures, &max_no_failures, &

```

```

374         mean_wagen_arrival, &std_wagen_arrival, &min_machine_repair_time, &
375         max_machine_repair_time, &end_warmup_time, &end_simulation_time);
376     fclose(infile);
377
378     if ((infile = fopen(inputfile_time, "r")) == NULL) {
379         printf("Could not open file %s\n", inputfile_time);
380     }
381     printf(" %d %d %d %d %f %f %f %f %f %f\n", number_of_machines,
382         min_productivity, min_no_failures, max_no_failures, mean_wagen_arrival,
383         std_wagen_arrival, min_machine_repair_time, max_machine_repair_time,
384         end_warmup_time, end_simulation_time);
385
386     int counter = 1;
387     while (!feof(infile)) {
388         fscanf(infile, "%f %d %f", &transfer_time[counter], &queue_size[counter], &
389             work_time[counter]);
390         printf("%f %d %f\n", transfer_time[counter], queue_size[counter], work_time[
391             counter]);
392         counter++;
393     }
394     fclose(infile);
395 }
396
397 void end_warmup()
398 {
399     sampst(0.0, 0);
400     timest(0.0, 0);
401     skaut_throughput = 0;
402 }
403
404 void report()
405 {
406     int i;
407     printf("\n*****\n");
408     printf("Report for %d number of failures per day\n", failure_nr);
409     printf("-----\nMachine load\n-----\n");
410     for (i=1; i <= number_of_machines; i++) {
411         printf("Machine %d\t", i);
412     }
413     printf("\n");
414     for (i=1; i <= number_of_machines; i++) {
415         printf("%f\t", filest(i));
416     }
417     printf("\n\n");
418
419     printf("-----\nAverage delay in queues\n
420     -----\n");
421     for (i=1; i <= number_of_machines; i++) {
422         printf("Queue %d \t", i);
423     }
424     printf("\n");
425
426     for (i=1; i <= number_of_machines; i++) {
427         printf("%f\t", sampst(0.0, -i));
428     }
429     printf("\n\n");
430     printf("Average queue delay: %f\n", sampst(0.0, -sampst_delays));
431     printf("System throughput: %d\n", skaut_throughput);
432     printf("Average throughput time: %f\n", sampst(0.0, -throughput_time));
433     printf("Min throughput time: %f\n", transfer[4]);
434 }
435
436 void push_array() {
437     memcpy(temp_transfer, transfer, TRANSFER_ARRAY_LENGTH*sizeof(float));

```

```

436 }
437
438 void pop_array() {
439     memcpy(transfer, temp_transfer, TRANSFER_ARRAY_LENGTH*sizeof(float));
440 }
441
442 void create_machine_fail_events(int n) {
443     int i;
444     float a[20];
445     memset(a, 0, 20*sizeof(float));
446     float span = (float)(end_simulation_time - end_warmup_time) / (float) n
447                 + 1.0; //max time between machine failures
448     float current_span = 0.0;
449     int machine;
450     float repair_time;
451     float breakdown_time;
452     for (i = 0; i < n; i++) {
453         current_span += span;
454         machine = (int)unirand(1, number_of_machines + 1, stream);
455         breakdown_time = unirand(0.0, current_span, stream);
456         repair_time = (5.0 + expon(log(max_machine_repair_time -
457                                     min_machine_repair_time), stream)) * 60.0;
458         if (a[machine] < breakdown_time) { //
459             a[machine] = breakdown_time + repair_time;
460         }
461         else { // if breakdown time clashes with the same machine then let the
462             // breakdown happen after the machine goes up again
463             breakdown_time = a[machine] + 1.0;
464             a[machine] = breakdown_time + repair_time;
465         }
466         transfer[3] = repair_time;
467         transfer[4] = (float)machine;
468         event_schedule(breakdown_time + end_warmup_time, EVENT_MACHINE_FAILURE);
469     }
470 }
471
472 void machine_failure() {
473     float repair_time = transfer[3];
474     int machine = (int)transfer[4];
475     machine_broken[machine] = repair_time;
476     // printf(" Machine %d broke down and it takes %f to repair\n", machine,
477     // repair_time/60.0);
478     event_schedule(sim_time + repair_time, EVENT_MACHINE_FIXED);
479 }
480
481 void machine_fixed() {
482     int machine = (int)transfer[4];
483     machine_broken[machine] = 0.0;
484 }

```

5.2 Inntaksgögn líkans

```

1 7 404 3 10 30.0 2.0 5.0 180.0 1000.0 58600.0
2 num min min max mean std min max warmup simulationtime
3 prod- no no wagen wagen repair repair
4 uction fail- fail- arrival arrival time time
5 ures ures

```

```

1 129.83 14 70.0
2 122.82 17 21.79
3 18.98 0 12.70
4 22.74 1 67.41
5 0.1 1 69.75
6 0.1 0 81.85
7 0.1 2 70.33

```

5.3 Keyrsluskýrslur