Hermun skautaskála hjá Ísáli

Gunnarr Baldursson & Ragnar Gísli Ólafsson

Apríl 2011

Útdráttur

Ble! Abstract

Efnisyfirlit

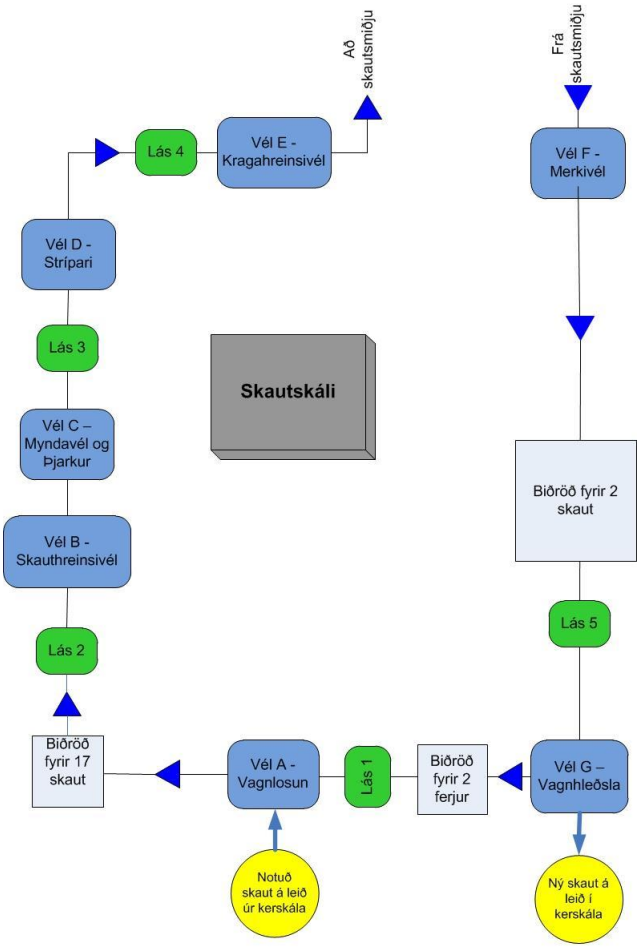
1 Inngangur

Alcan á Íslandi hf., betur þekkt sem Ísál, er hluti af Rio Tinto Alcan, fjölþjóðlegu fyrirtæki sem er stærsti álframleiðandi í heimi. Ísal 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, málarar, skrifstofufólk, bifvélavirkjar, viðskiptafræðingar, múrarar, matreiðslumenn, rafeindavirkjar, smiðir o.fl

Ísal 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 brunnu 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 skautskala 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 vakta 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 brunnin 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 steypt 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 vélr 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 handvirkt. 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 gír.

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, Líkan og Aðferðir

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

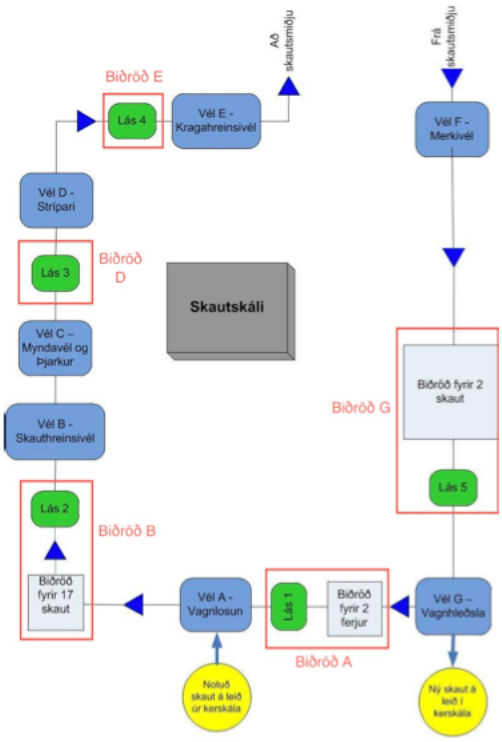
3.1 Umskipting skauta og afköst Skautskála

- 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 keri 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 eru það lágmarksafköst skautskála.
- Eftir straumhækkun þarf að skipta um skaut í öllum skálum á 26 daga fresti. Það gerir $\frac{24 \cdot 160}{26} + \frac{24 \cdot 160}{26} + \frac{24 \cdot 160}{26} = 443.07$ 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 veldisdreyfðir 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 þann 21. mars 2011 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 hálftíma 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.

4 Sannreyning Líkans

Heimildir

[1] Starfsmaður Ísal, *HermunIsal_2011_r2.pdf*. 2011.

[2] Starfsmaður Ísal, *Millitimar.xls*. 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
39 //#define LOADING_TIME_PER_SKAUT
40
41 // Global variables
42 int number_of_machines, min_productivity, min_no_failures, max_no_failures,
       skaut_throughput;
43 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;
44
45
46 int sampst_delays, throughput_time; // variable for queue delays and throughput
       time
47
48 int skaut_id, stream;
49 int queue_size[NUM_MACHINES +1];
50 float machine_broken[NUM_MACHINES +1];
51
52 int is_machine_busy[NUM_MACHINES +1],
53         queue_size[NUM_MACHINES +1];
54
55 float work_time[NUM_MACHINES + 1],
56         transfer_time[NUM_MACHINES +1]; // +1 is the less preferable simlib
               indexing scheme
57
58 float temp_transfer[TRANSFER_ARRAY_LENGTH];
59
60 FILE *infile, *outfile;
61
62 /* Function signatures */
63
64 // Usage: create_machine_fail_events(number_of_failures)
65 // Pre:   init_twister must be called for random number generation
66 // Post:  scheduled events have been created for machines
67 void create_machine_fail_events(int);
68
69
70 // Usage: push_array();
71 // Pre:   we expect that correct values are in transfer array
72 // Post:  our temp_transfer array now has the values in transfer_array
73 void push_array();
74
75 // Usage: pop_array();
76 // Pre:   we expect that correct values are in transfer_temp array
77 // Post:  our transfer array now has the values in transfer_temp
78 void pop_array();
79
80 // Usage: wagen_arrival();
81 // Pre:   EVENT_WAGEN_UNLOAD_ARRIVAL is the next event to be processed
82 // Post:  14 EVENT_SKAUT_ARRIVAL events are next to be processed on the event
       list.
83 void wagen_unload_arrival();
84
```

```
85 // Usage: skaut_arrival();
86 // Pre:   EVENT_SKAUT_ARRIVAL is the next event to be processed
87 // Post:  a skaut has been processed by a machine or put in it's queue.
88 //      subsequent events may have been scheduled
89 void skaut_arrival();
90
91 // Usage: skaut_departure();
92 // Pre:   EVENT_SKAUT_DEPARTURE is the next event to be processed
93 // Post:
94 void skaut_departure(); // do we need an event for departure?
95
96 // Usage: machine_failure();
97 // Pre:   EVENT_MACHINE_FAILURE is the next event to be processed
98 // Post:
99 void machine_failure();
100
101 // Usage: machine_fixed();
102 // Pre:   EVENT_MACHINE_FIXED is the next event to be processed
103 // Post:
104 void machine_fixed();
105
106 // Usage: end_warmup();
107 // Post:  SIMLIB statistical variables have been cleared
108 void end_warmup();
109
110 // Usage: parse_input(input_filename_data,input_filename_time);
111 // Pre:   input_filename_data,input_filename_time of type char[],
112 //      global variables from the input file exist.
113 // Post:  the global variables were assigned values from input_filename,
114 //
115 void parse_input(char[] ,char[]);
116
117 // Usage: x = N(muy, sigma, stream);
118 // Pre:   muy and sigma are of type float
119 //      stream is of type int
120 // Post:  x is a random gaussian distributed variable of type float
121 //      with mean muy and std sigma
122 float N(float muy, float sigma, int stream);
123
124 // Usage: report("the_report.out");
125 // Pre:   the values to be reported have values
126 // Post:  a report on program values and simlib statistics
127 //      have been APPENDED to "the_report.out"
128 void report();
129
130 // Usage: schedule_failures(i);
131 // Pre:   the global variable end_simulation_time has a value, i is of type int
132 // Post:  i failures have been scheduled uniformly on machines
133 //      with ?random? repair times on the interval [min_machine_repair_time,...
    max_machine_repair_time]
134 //      uniformly distributed over the interval 0...end_simulation_time
135 void schedule_failures(int i);
136
137 void queue_is_full();
138
139 int main()
140 {
141 // load datafiles
142         parse_input("adal_inntak.in","velar_og_bidradir.in");
143
144         // initialize arrays and variables
145         memset( is_machine_busy,0, NUM_MACHINES +1 );
146         memset( machine_broken,0, NUM_MACHINES +1);
147         skaut_throughput = 0;
148         sampst_delays = number_of_machines +1;
149         throughput_time = number_of_machines +2;
150
151         stream = 8;
152
153         int b;
```

```
154         for (b=1; b <= number_of_machines; b++) {
155                 printf("transfer_time[%d] = %f\n", b,transfer_time[b] );
156                 printf("busy %d broken %f \n",is_machine_busy[b],machine_broken
                        [b]);
157         }
158         // We perform simulation for "a few" failures per day
159         int i;
160         for (i = min_no_failures; i < max_no_failures; i++) {
161                 //for ( i=1; i<2; i++) {
162                 skaut_id = 1;
163                 skaut_throughput = 0;
164
165                 // Initialize rndlib
166                 init_twister();
167
168                 // Initialize simlib
169                 init_simlib();
170
171                 maxatr = 6; // how many attributes do we need?
172
173                 /* Schedule machine breakdown time */
174                 create_machine_fail_events(i);
175
176                 /* Schedule first wagen arrival */
177                 //transfer[3] = 1.0;
178                 event_schedule( 1.0, EVENT_WAGEN_UNLOAD_ARRIVAL );
179
180                 /* Schedule end of warmup time */
181                 event_schedule( end_warmup_time, EVENT_END_WARMUP );
182
183                 /* Schedule simulation termination */
184                 event_schedule( end_simulation_time, EVENT_END_SIMULATION );
185
186
187
188
189
190                 while (next_event_type != EVENT_END_SIMULATION) {
191
192                         timing();
193                         printf("event_type = %d, transfer[3] = %f\n",
                                next_event_type, transfer[3]);
194                         int k;
195                         for (k = 1; k <= number_of_machines; k++)
196                                 printf("Items in machines/queues %d:  %d, %d\n"
                                        , k, list_size[k], list_size[
                                        number_of_machines +k]);
197                         printf("\n");
198
199
200
201                         switch (next_event_type) {
202                         case EVENT_WAGEN_UNLOAD_ARRIVAL:
203                                 wagen_unload_arrival();
204                                 break;
205                         case EVENT_SKAUT_ARRIVAL:
206                                 skaut_arrival();
207                                 break;
208                         case EVENT_SKAUT_DEPARTURE:
209                                 skaut_departure();
210                                 break;
211                         case EVENT_MACHINE_FAILURE:
212                                 machine_failure();
213                                 break;
214                         case EVENT_MACHINE_FIXED:
215                                 machine_fixed();
216                                 break;
217                         case EVENT_END_WARMUP:
218                                 end_warmup();
219                                 break;
```

```
220                         case EVENT_END_SIMULATION:
221                                 report();
222                                 break;
223                         }
224                 }
225         }
226 }
227
228 void wagen_unload_arrival()
229 {
230
231         int i;
232         int current_unit = 0;
233         for (i = NUM_MACHINES; i>0; i--) {  //add delay if machine is broken or
                there is a broken machine before current one
234                 if (machine_broken[i] > 0.0) {
235                         event_schedule(sim_time + machine_broken[i],
                            EVENT_WAGEN_UNLOAD_ARRIVAL);
236                         return;
237                 }
238         }
239
240         for (i=1; i <= WAGEN_LOAD; i ++) {
241
242                 transfer[3]=1.0;
243                 transfer[4] = sim_time + (i * 0.01); // skaut entering system
                        time
244                 transfer[6] = (float) skaut_id++;
245                 //printf("tr4 in wagen: %f\n", transfer[4]);
246                 event_schedule( sim_time + ( i* 0.01), EVENT_SKAUT_ARRIVAL);
247         }
248
249         float wagen_arrival_zeit = sim_time + 30.0 * 60.0; // this should be
                sampled from a distribution!!
250         event_schedule(wagen_arrival_zeit, EVENT_WAGEN_UNLOAD_ARRIVAL);
251 }
252
253
254 void skaut_arrival()
255 {
256         push_array();
257         int current_unit = (int)transfer[3];
258         if (machine_broken[current_unit] > 0.0) {
259                 if (list_size[number_of_machines + current_unit] < queue_size[
                        current_unit] || queue_size[current_unit] == 0) {  // if
                        current machine is broken then delay it.
260                         event_schedule(sim_time + machine_broken[current_unit]
                                + work_time[current_unit], EVENT_SKAUT_ARRIVAL);
261                         return;
262                 }
263         }
264         int i;
265         for (i = NUM_MACHINES; i>current_unit; i--) {  //add delay if there is
                a broken machine before current one
266                 if (machine_broken[i] > 0.0) {
267                         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
268                                 event_schedule(sim_time + work_time[
                                        current_unit + 1]+  machine_broken[i],
                                        EVENT_SKAUT_ARRIVAL); //also if next queue
                                        is full then delay it.
269                                 return;
270                         }
271                 }
272         }
273
274
275
```

```
276         // check if machine is not busy
277         if (list_size[current_unit] == 0 && machine_broken[current_unit] ==
                0.0) {
278                 sampst(0.0, sampst_delays);
279                 sampst(0.0, current_unit);
280
281                 list_file(FIRST, current_unit); // last := first here because
                        there are only to be 0 or 1 items in machine
282
283                 // schedule departure after machine processing time
284                 pop_array();
285                 event_schedule(sim_time + work_time[current_unit],
                        EVENT_SKAUT_DEPARTURE);
286         } else {
287
288                 if (list_size[number_of_machines + current_unit] == queue_size[
                        current_unit]) {
289                         int i;
290                         for (i = 1; i<10; i++) {  //add delay if machine is
                                broken or there is a broken machine before current
                                one
291                                 printf(" %f, limit \n",machine_broken[i]);
292                         }
293                         printf("BOOM! UNIT %d exploded with %d items!\n",
                                current_unit, list_size[number_of_machines +
                                current_unit]);
294                         exit(1);
295                 } else {
296                         transfer[5] = sim_time;
297                         list_file(LAST, number_of_machines + current_unit);
298                         //printf("puting skaut in queue: %d\n", current_unit);
299                 }
300
301         }
302
303 }
304
305 void skaut_departure()
306 {
307         push_array();
308         int current_unit = (int) transfer[3];
309         int i = 0;
310         for (i = NUM_MACHINES; i>=current_unit; i--) {  //add delay if machine
                is broken or there is a broken machine before current one
311                 if (machine_broken[i] > 0.0) {
312                         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.
313                                 event_schedule(sim_time + work_time[
                                        current_unit + 1]+  machine_broken[i],
                                        EVENT_SKAUT_DEPARTURE); //also if next
                                        queue is full then delay it.
314                                 return;
315                         }
316                         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]);
317                         break;
318                 }
319         }
320
321         if (current_unit == MACHINES_ON_THE_LEFT_SIDE) {
322                 skaut_throughput += 2;
323                 sampst(sim_time - transfer[4], throughput_time);
324                 list_remove(FIRST,current_unit);
325         } else {
326                 list_remove(FIRST,current_unit);
327                 pop_array();
328                 transfer[3]++;
```

```
329                     event_schedule(sim_time + transfer_time[(int)(transfer[3])-1],
                            EVENT_SKAUT_ARRIVAL);
330             }
331
332
333             if (list_size[number_of_machines + current_unit] != 0) {
334                     pop_array();
335
336                     list_file(FIRST,current_unit); // first equals last because
                            size should only be 1
337                     pop_array();
338
339                     list_remove(FIRST, number_of_machines + current_unit);
340                     pop_array();
341
342                     sampst(sim_time - transfer[5], sampst_delays);
343                     sampst(sim_time - transfer[5], current_unit);
344                     event_schedule(sim_time + work_time[current_unit],
                            EVENT_SKAUT_DEPARTURE);
345             }
346     }
347
348
349     void parse_input(char inputfile_data[], char inputfile_time[])
350     {
351
352
353             if ((infile = fopen (inputfile_data, "r")) == NULL) {
354                     printf("Could not open file %s\n",inputfile_data);
355             }
356
357             fscanf (infile, "%d %d %d %d %f %f %f %f %f %f", &number_of_machines, &
                    min_productivity, &min_no_failures, &max_no_failures, &
                    mean_wagen_arrival, &std_wagen_arrival,  &min_machine_repair_time,
                    &max_machine_repair_time, &end_warmup_time, &end_simulation_time);
358             fclose(infile);
359
360
361             if ((infile = fopen (inputfile_time, "r")) == NULL) {
362                     printf("Could not open file %s\n",inputfile_time);
363             }
364             printf( "%d %d %d %d %f %f %f %f %f %f\n", number_of_machines,
                    min_productivity, min_no_failures, max_no_failures,
                    mean_wagen_arrival, std_wagen_arrival,  min_machine_repair_time,
                    max_machine_repair_time, end_warmup_time, end_simulation_time);
365
366             int counter = 1;
367             while (!feof(infile)) {
368                     fscanf(infile, "%f %d %f", &transfer_time[counter], &queue_size
                            [counter], &work_time[counter] );
369                     printf("%f %d %f\n", transfer_time[counter], queue_size[counter
                            ], work_time[counter] );
370                     counter++;
371             }
372             fclose(infile);
373
374     }
375
376     void end_warmup()
377     {
378             sampst(0.0, 0);
379             timest(0.0, 0);
380             skaut_throughput = 0;
381     }
382
383     void report()
384     {
385             printf("System throughput: %d\n", skaut_throughput );
386             int i;
387             for (i=1; i <= number_of_machines; i++) {
```

```
388                     printf("Machine %d: %f\n", i, filest(i) );
389             }
390             for (i=1; i <= number_of_machines; i++) {
391                     printf("Avg delay in queue %d: %f\n", i, sampst(0.0, -i));
392             }
393             printf("Avarage queue delay: %f\n", sampst(0.0, -sampst_delays));
394
395             printf("Average throughput time: %f\n", sampst(0.0, -throughput_time));
396             printf("Min throughput time: %f\n", transfer[4]);
397
398     }
399
400     void push_array() {
401
402             memcpy(temp_transfer,transfer,TRANSFER_ARRAY_LENGTH*sizeof(float));
403     }
404
405     void pop_array() {
406             memcpy(transfer,temp_transfer,TRANSFER_ARRAY_LENGTH*sizeof(float));
407     }
408
409     void create_machine_fail_events(int n) {
410             int i;
411             float a[20];
412             memset(a,0,20*sizeof(float));
413             float span = (float)(end_simulation_time - end_warmup_time) / (float) n
                    ;  //max time between machine failures
414             float current_span = 0.0;
415             int machine;
416             float repair_time ;
417             float breakdown_time;
418             for (i = 0;i<n;i++) {
419                     current_span+=span;
420                     machine = (int)unirand(1,number_of_machines+1,stream);
421                     breakdown_time = unirand(0.0,current_span,stream);
422                     repair_time =(5.0 + expon(log(max_machine_repair_time -
                            min_machine_repair_time),stream))*60.0;
423                     if (a[machine]<breakdown_time) {  //
424                             a[machine] = breakdown_time+repair_time;
425                     }
426                     else { // if breakdown_time clashes with the same machine then
                            let the breakdown happen after the machine goes up again
427                             breakdown_time = a[machine] + 1.0;
428                             a[machine] = breakdown_time+repair_time;
429                     }
430     <<<<<<< HEAD
431                     //printf("Span from 0.0 to %f.  Machine %d broke down at time %
                            f and it takes %f to repair\n", current_span, machine,
                            breakdown_time, repair_time/60.0);
432     =======
433     >>>>>>> 3ead7a5c7134ea5eb9fbb3c5b8fa989027be99a5
434                     transfer[3] = repair_time;
435                     transfer[4] = (float)machine;
436                     event_schedule(breakdown_time, EVENT_MACHINE_FAILURE );
437             }
438     }
439
440     void machine_failure(){
441             float repair_time = transfer[3];
442             int   machine     = (int)transfer[4];
443             machine_broken[machine] = repair_time;
444             printf(" Machine %d broke down and it takes %f to repair\n", machine,
                    repair_time/60.0);
445
446             event_schedule(sim_time + repair_time, EVENT_MACHINE_FIXED);
447     }
448
449     void machine_fixed(){
450
451             int   machine     = (int)transfer[4];
```

```
452             machine_broken[machine] = 0.0;
453         }
```

5.2 Inntaksgögn líkans

5.3 Keyrsluskýrslur